PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 4 : F41G 5/08, 3/14

A1

(11) International Publication Number: WO 88/ 02841

(43) International Publication Date: 21 April 1988 (21.04.88)

(21) International Application Number: PCT/US87/02435

(22) International Filing Date: 24 September 1987 (24.09.87)

(31) Priority Application Number: 921,164

(32) Priority Date: 17 October 1986 (17.10.86)

(33) Priority Country: US

(71) Applicant: HUGHES AIRCRAFT COMPANY [US/US]; 7200 Hughes Terrace, Los Angeles, CA 90045-0066 (US).

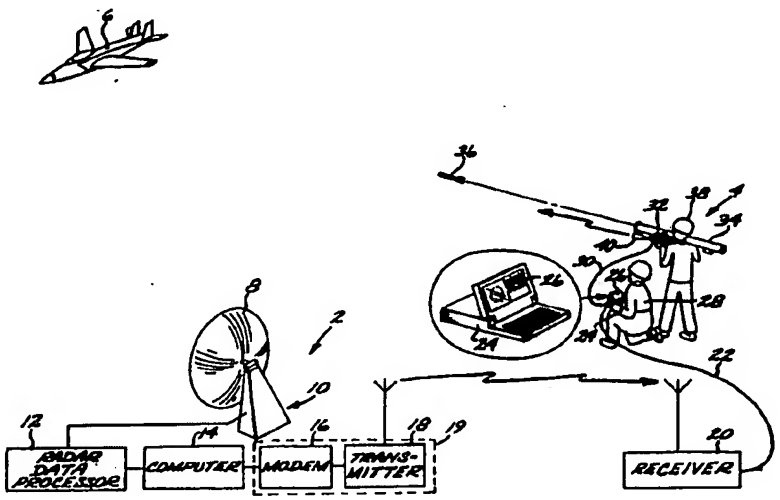
(72) Inventors: HERGESHEIMER, Peter, D. ; 1001 Bonnie Ann Court, La Habra, CA 90631 (US). JENSEN, John, Neil ; 355 South Los Robles, Apt. 240, Pasadena, CA 91101 (US).

(74) Agents: RUNK, Thomas, A. et al.; Hughes Aircraft Company, P.O. Box 45066, Bldg. C1, M/S A126, Los Angeles, CA 90045-0066 (US).

(81) Designated States: DK, NL, NO.

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: WEAPON AUTOMATIC ALERTING AND CUEING SYSTEM



(57) Abstract

A system for alerting and cueing a weapon system (34) to detected targets, the system including a sensor (2) for providing target data to a computer (24) and for providing weapon system (34) location and pointing data to the computer (24) for calculation of the relative directions and ranges of the targets from the computer (24). The weapon system (34) is remotely located from the sensor (2) and may be a portable launcher. In the case of a particular target (6) being designated, the computer (24) calculates the errors between the pointing direction of the weapon system (34) and the location of the designated target and communicates these errors to the weapon system (34) sight (32) for guidance of the weapon system to the designated target (6). The computer (24) has multiple, selectable displays one of which is a situation display (78) for presenting the relative directions and certain characteristics of the detected targets and another of which is a parameter display (80) for displaying numerical data relating to a selected target, and in one embodiment, the computer (24) may be a portable computer.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	FR	France	ML	Mali
AU	Australia	GA	Gabon	MR	Mauritania
BB	Barbados	GB	United Kingdom	MW	Malawi
BE	Belgium	HU	Hungary	NL	Netherlands
BG	Bulgaria	IT	Italy	NO	Norway
BJ	Benin	JP	Japan	RO	Romania
BR	Brazil	KP	Democratic People's Republic of Korea	SD	Sudan
CF	Central African Republic			SE	Sweden
CG	Congo	KR	Republic of Korea	SN	Senegal
CH	Switzerland	LI	Liechtenstein	SU	Soviet Union
CM	Cameroon	LK	Sri Lanka	TD	Chad
DE	Germany, Federal Republic of	LU	Luxembourg	TG	Togo
DK	Denmark	MC	Monaco	US	United States of America
FI	Finland	MG	Madagascar		

WEAPON AUTOMATIC ALERTING AND CUEING SYSTEM

BACKGROUND OF THE INVENTION

Air defense weapon systems have been configured in the presence of many conflicting requirements and environmental factors. On the one hand, it is desirable that a given emplacement or unit of air defense weaponry be small, relatively inexpensive and have a limited suite of sensors in order to maximize the number of units deployed and reduce their value as a target. On the other hand, the increasingly sophisticated aircraft and missiles that are the targets for such weapon systems have presented more and more difficult acquisition and tracking problems for the relatively simple engagement systems that are part of such air defense weapons.

The development of handheld or so-called "manpad" weapon systems has substantially improved the capability of such small and easily deployed air defense weapons. Weapons such as the Stinger missile, which is a heat seeking manpad launched guided missile, had dramatically improved the air defense capabilities of surface-based military units. However, there still remain the extremely difficult acquisition and targeting issues for such weapons. Absent some sort of warning, cueing and ranging capability, manpad weapons may not be brought to bear on the target with sufficient time and launch window to provide for efficient engagement. Furthermore, such manpad weapons are severely limited under adverse weather conditions due to the requirement that the target be visually acquired. The result of these problems is either a failure to launch the missile or a miss due to failure to acquire or launch when the target is within the proper range. While personnel training can improve performance of manpad weapon systems, there still remains a serious problem with regard to the consistency of

performance and effectiveness of such weapons against a variety of targets. The addition of new sensors to such manpad weapons to overcome some of these limitations would either render them inappropriate for manpad use or complicate their use and deployment and increase their cost beyond acceptable limits for their point defense mission. There is, therefore, a requirement for means to inexpensively augment the target acquisition, tracking and firing capabilities of manpad weapons.

SUMMARY OF THE INVENTION

The present invention provides a means of economically, reliably and efficiently presenting target acquisition, ranging and firing information to manpad weapon systems, such as the ones commonly used for air defense. The invention accomplishes these desirable functions by processing information from various sensors, correlating target information with weapon system location and thereafter providing refined data to the weapon system that allows for rapid acquisition, tracking and firing of the weapon by the human operator using a modified version of the normally supplied visual sight display. The invention thereby provides capabilities to relatively unsophisticated and inexpensive manpad weapon systems that become the equivalent to a co-location of sophisticated sensors with the weapon system. This desirable capability is achieved without the cost, complexity and difficulty of deployment associated with more elaborate point defense systems incorporating co-located sensors. The invention uses a novel combination of radar target acquisition and tracking, optical tracking and infra-red homing when used in combination with heat seeking missiles such as the Stinger manpad weapon system. Previous weapon systems that use radar for acquisition and tracking have had co-located sensors and presented tracking information to a visual situation display, if one was used. The present invention uses the human

operator as a means of completing the tracking problem, whether the missile or other weapon has self-contained tracking or is guided by the human operator through his optical sight. Thus, the invention combines the best features of radar, which provides relatively long acquisition ranges and velocity information, with the decision making and tracking capabilities of the human operator when the target comes within visual range.

The invention further provides a beneficial effect in that it is unnecessary to locate a central major computer system near either the radar or weapon system, thereby further dissipating the resources of the system and avoiding the creation of a high value target. Such a distributed weapon system also has the benefit that a loss of one or more weapon platforms or sensors does not render the system incapable of continued operation. Such a capability is particularly important in a modern battlefield scenario and especially considering the "one shot" nature of many of the weapons that would be controlled. The invention also can provide for displays to weapon system managers that indicate performance and status in real time during an engagement. While the system has been configured with a small computer terminal co-located with the weapon, it is also possible to use other configurations of distributed data processing via co-location of portions of the data processing resources with the sensor systems, intermediate communications, command and control functions and the manpad weapon. Such a distribution may be desirable in order to improve survivability of the invention in the hostile environments encountered in battle.

The present invention utilizes the beneficial arrangement of well known components and components specifically developed for the invention to provide heretofore unavailable capabilities in an air defense system. One of the components used is a sensor or plurality of sensors providing input data describing to the range, azimuth, elevation and signature of

targets in the airborne environment surrounding the sensor. One type of sensor system that provides such information is a three-dimensional radar of the type commonly used for battlefield surveillance. For example, a Low Altitude Surveillance three dimensional Radar (LASR), for example, a derivative of the AN/TPQ-36 artillery locating radar, provides target range, azimuth and elevation information to the system in map coordinates. Information derived from such a 3D radar includes signature information associated with returns from a target that allows the identification of targets as either helicopters or conventional aircraft. The information derived from the LASR may then be transmitted over a conventional digital radio link typified by the military Joint Tactical Information Distribution System (JTIDS) currently in use by the United States Military. This information may be directed to a communications and command station which can include a computer capable of correlating the target information and the known locations of the manpad weapons system and providing outputs of refined data that may be displayed at the manpad subsystem for the purpose of acquisition and firing. An alternative preferred embodiment of the invention discussed herein can provide for the direct communication from the LASR to the manpad subsystem provided that a portable computer terminal capable of performing these calculations is collocated with the manpad weapon. Such computer terminals are readily available in militarized versions suitable for this purpose. In either case, the output of the computer is used for two purposes: to drive the visual acquisition and tracking sight of the weapon launcher and, to provide a situation display to the fire unit commander to indicate the location and status of targets in the vicinity of the fire unit. While it is not necessary that a fire unit consist of two individuals, one performing the tasks of tracking a target and launching the missile, and another performing the tasks of monitoring and designating the target displayed in the

situation display, it has been found that such a division of duties provides for an efficient operation of a fire unit.

An important element in the transition from the information supplied by the sensor system to the visual tracking by the human operator of the target through the viewfinder of the weapon is the means of deriving azimuth and elevation of the weapon sight in relationship to the earth coordinates. The present invention utilizes a combination of inclinometer and fluxgate compass to provide this information. The precision fluxgate compass developed from commercially available units for the invention may be compensated for nearby magnetic fields by a simple rotation through 360 degrees, the resultant output being fed to the computer and normalized for the earth's magnetic variation as part of the data base in the computer program. Thus, the present invention provides for rapid calibration of the sensors detecting the inclination and azimuth of the weapon launcher. A digital output of those sensors is fed to the computer and a signal comparing these signals with the desired azimuth and elevation for acquisition is sent to the viewfinder for comparison by the human operator. The computer thus accepts inputs of target X, Y, Z and $\dot{X}$, $\dot{Y}$, $\dot{Z}$ in the chosden reference system and converts them to range, azimuth and elevation and displays them in a maplike display in one portion of the computer screen while another portion of the computer screen displays numerical information relevant to the parameters for the target selected. The situation display also contains a readout of the azimuth and elevation of the weapon, thereby providing the fire unit commander with an instantaneous view of the status of acquisition and tracking.

The visual display provided to the weapon launcher is a modified version of the Stinger peripheral ranging system described in copending United States patent application for a Peripheral Vision Guidance Display having a serial number of 06/673,257 and incorporated by reference herein. In the modified version of this concept utilized in the present invention,

this peripheral vision guidance system incorporates a ring of light emitting diodes (LEDs) to indicate the directions in which the axis of the missile launcher must be guided to acquire the target. In the version of the display used in the present invention, two sets of LEDs are used, one red and one green. The red LEDs correspond to the azimuth and elevation commanded and the green LEDs correspond to an inrange system. In practice, a blinking red LED indicates that the weapon should be moved in that direction and when the red LEDs become steady, the axis of the launcher is pointed at the target. When the green LEDs are off the target is not yet in range. The green LEDs come on when the target is in firing range.

By use of the invention, the operator may acquire and track a target in conditions of low visibility rather than if only visual sighting means were used where low visibility poses substantial problems, thereby extending and improving the operational capabilities of the weapon. Thus, the current invention provides for an efficient and rapid acquisition of targets by a two man team in a battlefield environment without the requirement for collocation of sensors or elaborate resources. Furthermore, the invention utilizes simple and economic sighting and sensor means on the launcher that may be disconnected from the reusable launch tube and used a number of times without modification or refurbishment. The invention, therefore, provides for a higher level of performance relevant to the training required without a significant increase in the cost of each fire unit, the overall cost of the system and its vulnerability. In practice, it has been found that the present invention provides a steering system so simple to use that first time untrained operators can steer to within 1 degree of the correct azimuth and elevation in less than 20 seconds. Trained operators can regularly accomplish this accuracy in less than 4 seconds. The fire unit may also be provided with an Identification, Friend and Foe (IFF) sensor that can prevent an

unwanted launch against a friendly target. Such an IFF sensor has the capability to electronically interrogate a target on the bore sight of the target and receive an encoded reply from a friendly aircraft, thereby providing a position warning to the operator that the launch should not occur.

It can be seen from the above discussion of the invention that it provides hitherto unavailable capabilities in an air defense weapons system without unnecessary sophistication or expense. The use of proven and available components in a previously unexplored arrangement thus provides many of the features of complex and expensive air defense systems while retaining the relative simplicity and low vulnerability of manpad systems. Other features and advantages of the present invention will become apparent from the following detailed description taken in conjunction with the accompanying drawings which illustrate, by way of example, the principles of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a perspective of the invention deployed in a battlefield illustrating the relationship of the major components.

Figure 2 is a block diagram of the major components associated with the computer terminal illustrating the primary interfaces with the sensor system and the weapon launcher.

Figure 3 is an illustration of a display on the computer terminal pictorially describing an engagement in which various targets are presented and the error signals between the launcher pointing vector and the command vectors are indicated.

Figure 4 is an illustration of a parameter menu available at the computer terminal for operator selection of parameters to be displayed.

Figure 5 is a perspective of the sight employed with

the weapon launcher which provides cueing information to the operator.

Figure 6 is a view of the sight taken along the optical bore illustrating the cueing light emitting diodes used to provide information regarding azimuth, elevation and range of the target.

DETAILED DESCRIPTION OF THE INVENTION

As shown in the exemplary drawings, the present invention is embodied in a weapon automatic alerting and cueing system which uses a remote sensor system 2 to provide information to a remote weapon launcher team 4 that allows for the designation, acquisition and engagement of targets 6 that are encountered in a battlefield environment. The invention provides a means of utilizing handheld weapons such as the Stinger missle system in combination with a remotely located sensor 2, thereby providing many of the advantages of a complex and sophisticated air defense system while diminishing the value of the individual components of the weapon system due to their distribution and relatively lower cost and sophistication. In the exemplary embodiment discussed herein, certain components of the system have been chosen for their proven performance and availability but it should be noted that the invention may also be configured with components developed for the specific purpose of enhancing the various features of the invention.

As illustrated in FIG. 1, the invention utilizes a remote sensor system 2, in this case incorporating a well known 3D radar 8 that is capable of deriving information about the target 6 in azimuth, elevation and range from the radar site 10. The azimuth, elevation and range information from the radar 8 is then transferred from the radar data processor to a special purpose computer 14 which calculates the position of target 6 with reference to a fixed reference plane. This

information is then transferred to a modem 16 which encodes the information for transmission by a transmitter 18 to the remote weapon site 4. Modem 16 and transmitter 18 are typically combined in a unit 19 such as JTIDs as described above. Receiver 20 receives the information for the targets acquired and tracked by radar 8 and transfers this information via cable 22 to a computer terminal 24 that includes display unit 26 available to weapon commander 28 at the weapon site. Weapon commander 28 utilizing the information displayed on display 26 analyzes the information presented and designates or "hooks" a target using the keyboard of computer terminal 24. After the target is hooked, parameters describing the range, azimuth and elevation of the target from the location of the weapon site 4 are transmitted via cable 30 to the sight 32 of weapon launcher 34, in this case a launcher for the Stinger missle 36. Weapon operator 38, sighting through sight 32 is provided with information indicating the direction in which the missle launcher must be pointed and an in-range or out of range message thereby drastically simplifying the acquisition of target 6. Tracking of target 6 by weapons launcher 34 is carried out in the conventional manner for such manpad weapons, by visual centering of the target in an optical aperture.

Thus, the present invention provides a means of acquisition and target designation which is a substantial improvement on previous methods of relying upon visual sightings by the weapon crew in a battlefield environment. The weapon station may also be provided with an identification - friend or foe (IFF) set 40 which can interrogate a target and identify a friendly target thereby avoiding a launch against a friendly vehicle.

While the present invention is illustrated in a configuration which routes sensor data directly to the fire unit, the invention may also use command centers and sensor data may be routed to the command centers and thence to the fire unit

or some combination thereof to enhance survivability. A three dimensional radar has been described as the basis for the sensor system, but an equivalent result may be obtained by the use of cooperative two dimensional sensors with the calculation of the X, Y, Z coordinates of the target in a fixed reference plane being performed by a computer similar to the one discussed above but which accepts the two dimensional track information from a plurality of sensors.

COMPUTER TERMINAL

Fig. 2 is a block diagram illustrating the relationship of the major elements of the software and hardware of computer terminal 24 at the weapon sight and its connections with cueing sight 32 and the external information coming from radar 8 (FIG. 1). The radar passes 3D radar data and time-of-day over the data link to receiver 20 and thence to the computer terminal 24. A computer found to be usable is the Compass 2 portable computer from Grid Systems Corporation, Mountain View, California. U.S. Patent 4,571,456 has been issued covering the features of this computer and is incorporated herein by reference. The data may be formatted as described in the following discussion of the radar track information data link, and is modulated with CCITT V.22 DPSK modulation. The data rate is 1200 BPS. The built-in modem 42 in the computer terminal demodulates the signal and passes the resulting 8-bit bytes to the Radar Message Processing module 44 which compiles and decodes the messages. Time-of-day messages are used to reinitialize the internal clock 46 in the computer terminal 24 to ensure that the computer terminal clock 46 does not drift from the radar clock. The time-of-day messages are transmitted from the radar approximately every 2 seconds.

Track messages are sent to the Parallax Correction module 48 whereby the data link reference point location (DLRP)

50 and the fire unit location 52 are used to adjust the North, East and altitude positions of the tracks so they are represented as offsets from the fire unit (Stinger) location. The track information is then used to update the track table in the Track Table Manager module 54. The Track Table Manager 54 will create a new table entry if the track numbers in the table do not match the track number in the received track information. The Track Table Manager 54 will also drop a track from the track table if the track has not been updated for a preselected period of time.

Target Parameter information is extracted from the track table for each target and sent to the Track Position Extrapolation module 56. This module uses the tracks' time tags and the current time-of-day to compute the age of the track. This data and the track velocities are used to determine how far the tracks have moved since the last radar update. By adding this new data to the original track positions, the current positions of the tracks are derived. These updated track positions are sent to the Air Situation Display module for display.

The Air Situation Display module 58 updates a graphics picture of the air situation at the fire unit displayed on the terminal display unit 26 (FIG. 1). The air situation display shows each track received from the radar relative to the position of the fire unit location. Each track is displayed with a preselected symbology that indicates if the track is friendly, hostile, or unknown and whether the track is a fixed or a rotary wing aircraft. The Air Situation Display module 58 initially displays the fire unit location in the center of the screen as a default, but allows the operator to move the display up, down, right, left or back to center on the display. The air situation display scaling may also be selected by the operator ranging from 10 by 10 kilometers to 160 by 160 kilometers in selectable steps. An alert circle radius which is operator

selected is also displayed on the air situation display. When a track enters the alert circle centered about the fire unit location, an audible alert may be generated by the computer terminal if the operator desires. The audible alert may be silenced by the operator until another track enters the alert circle.

The Air Situation Display module 58 provides the operator with four different methods for selecting or "hooking" a target. The first is with a "hook box" that may be positioned near the desired track by the operator to "capture" it. The second method is by selecting a priority level wherein the track with the corresponding priority level will be hooked. The third method is by sequencing from one track to the next. A fourth is by entering the track number of the desired track. The display indicates the hooked track by displaying the hook box around the target symbology. The hooked track is identified in the track table.

The air situation display also indicates the current position of the Launcher in azimuth and elevation. The azimuth is displayed as a line radiating from the fire unit location in the direction of the Stinger azimuth. The elevation is displayed on a vertical bar graph next to the air situation display. In addition, the same bar graph is used to display the elevation of the hooked target.

The software in the computer terminal has the ability to prioritize the tracks received from the radar based upon input from the operator. This function is performed by the Track Prioritization module 60. The operator may select a priority sector to aid in the track prioritization. This sector is displayed on the air situation display 58 as a pair of angled lines. The Track Prioritization module 60 uses the tracks' current range from the weapon site, the projected minimum range, velocity, altitude, degree of maneuvering, and identification to determine their respective priorities. The module sorts the

tracks in order of priority and assigns each a priority number. This number can be used to hook a track.

Information for the hooked target is passed from the Track Table 54 to the Target Position Extrapolation module 56. After each update from the Stinger pointing sensor, about every 0.1 second, the hooked track's position is calculated in the Track Position Extrapolation module 56. The updated position is sent to the Rectangular To Polar Conversion module 64 which converts the target's Easting, Northing and altitude position into slant range, azimuth and elevation position with respect to the fire unit location. This information is used for the weapon sight cueing, for display to the operator on the target metric display, and for calculating when the target is in range of the weapon.

The In-Range Calculation module 66 uses this updated target position, heading and velocity from the rectangular to Polar Conversion module 64 to determine if the target can be engaged by the Stinger weapon. If the target is not yet within range, the In-Range Calculation module 66 provides the time remaining before the target will be engaged. The in-range calculation results are passed to the weapon sight and to the target metric display.

Metric information on the hooked target is presented to the operator to aid in determining the engagability of the target. This function is accomplished by the Target Metric Display module 68. The module displays the time-to-engage which also indicates if the target is currently in range of the weapon or if the target will not become within range of the weapon based on its current heading and course. Other target information displayed is target range, azimuth, elevation, altitude, velocity, track number and priority level. The units for azimuth and elevation may be selected by the operator as degrees or mils. The target velocity units may also be selected to be meters per second or knots. The weapon control order for

the hooked target is also displayed. The computer terminal also displays the scale of the air situation display, the time-of-day and the status of the data link communications.

Weapon cueing is implemented using a feedback control loop involving a pointing sensor 70 located on the weapon, the computer terminal for sensor calibration and feedback error calculation, a special sight for displaying the pointing error, and a human operator for correcting the pointing error by positioning the weapon. The pointing sensor attached to the launcher is a digital compass/inclinometer. A particular sensor found to be usable in one embodiment of the invention is the Model 236 Precision heading/tiltsensor produced by Digicourse Inc., P.O. Box 50699, New Orleans, Louisiana 70150. The sensor senses the earth's magnetic field to determine the azimuth orientation of the weapon and senses the earth's gravity to determine the elevation orientation of the weapon. Weapon position messages from the pointing sensor are received at the computer terminal and processed by the Weapon Message Processing module 72. Correction for the individual sensor is then corrected by the Pointing Sensor Calibration module 74. This module adjusts for magnetic deviations affecting the pointing sensor and elevation biases in the pointing sensor. The corrected weapon position data is used by the Pointing Error Calculation module 76 to generate the pointing error information. The pointing error is determined by differencing the hooked target's azimuth and elevation with that of the weapon. If the pointing error is small enough, the module 74 sends an on-target indication to the Target Metric Display 68. The Weapon Message Processing module 72 converts the pointing error informaton into up, down, right and left commands. These commands combined with the in-range indication from the In-Range Calculation module 66 are sent to the weapon cueing sight 32. This special sight cues the weapon

operator to point up or down, left or right using four red blinking lights in this case red LEDs. Using these cueing lights, the weapon operator can position the weapon axis so that it is pointing at the target. The sight includes additional green LEDs to indicate when the target is within range of the weapon.

The modules referred to above may be implemented in software executable by the computer terminal 24. One such embodiment developed for the invention is included in the Appendix attached hereto and incorporated herein by reference.

RADAR TRACK INFORMATION DATA LINK

The data link which is used to broadcast radar track information from the LASR 3D radar to the computer terminal display units located at the weapon site is described below. Two message types are used; a track message for passing information for one of up to ten tracks to the weapon site and a time message for synchronizing the computer terminal clock 46 with the LASR clock. Each message is composed of 14 asynchronous characters. The characters are sent serially over the link at a rate of 1,200 BAUD with each character consisting of one start bit, one stop bit, eight data bits (LSB first), and one parity bit representing the even parity of the eight data bits. The serial bit stream is modulated by a Bell 212A compatible modem before being transmitted to the computer terminal display units. Each of the up to tens tracks sent over the data link is updated every radar antenna rotation (1.82 seconds for the candidate 3D radar). At the end of each 1.82 second rotation, a time message is also sent over the link. The computer 14 at the radar site 10 adjusts the time value in the time message so that it will be current when it is received at each computer terminal display. This function permits the clock

resident in each computer terminal to be synchronized with the LASR clock without any offsets resulting from communications delays. The time tags associated with each track update can then be used to accurately determine the track age between updates. Track age is used to extrapolate its current position based on the position of the last update and the track velocity.

While the information to be transferred between the sensor and the computer terminal may be formatted and transmitted in various ways, one advantageous form of such messages has been developed by the inventors and is tabulated in Tables 1 through 6 following. Those skilled in the art will recognize the efficiency and suitability of the formats developed but may find other advantageous methods of implementing this procedure.

TABLE 1
SUMMARY OF TRACK MESSAGE FORMAT

TRACK MESSAGE		RANGE	RESOLUTION
Sync Byte 1	8		
Sync Byte 2	8		
Message Type	2		
Spare	2		
Aircraft Type	1		
Track ID	2		
Track Number	8		
X-Position	16	+/-76,800 meters	2.34375 meters
Y-Position	16	+/-76,800 meters	2.34375 meters
Altitude	16	+/-76,800 meters	2.34375 meters
X-Velocity	8	+/-365 meters/sec	2.856 meters/sec
Y-Velocity	8	+/-365 meters/sec	2.856 meters/sec
Time Tag	8	15.9375 seconds	62.5 millisec
Checksum	8		
	112 bits = 14 bytes		

TABLE 2
CONTENTS DESCRIPTION - TRACK MESSAGE

Sync Byte #1	An 8-bit number used with Sync Byte #2 by the receive processor to determine the start of the message. It has a constant value of 'AA'H.
Sync Byte #2	An 8-bit number used with Sync Byte #1 by the receive processor to determine the start of the mesesage. It has a constant value of '55'H.
Message Type	A 2-bit word indicating the type of message. For this mesesage, the message type = '01'B.
Spare	All spare bits shall be set to zero.
Aircraft Type	A single bit indicating the aircraft type as follows: 0 Fixed Wing 1 Rotary Wing
Track ID	A 2-bit designation of the track's identification as follows: 00 Unknown 01 Designated Unknown 10 Friendly 11 Hostile A designated unknown track is a track of unknown identity that has been designated for special purposes.
Track Number	A 8-bit reference number identifying the track described by this message.
X-Position	A 16-bit 2's complement value for the distance in the computer terminal East direction of the track from the data link reference point. The LSB represents 2.34375 meters. The X-Position range is from -76,800.00000 meters to +76,797.65625 meters.

Y-Position	A 16-bit 2's complement value for the distance in the computer terminal North direction of the track from the data link reference point. The LSB represents 2.34375 meters. The Y-Position range is from -76,800.00000 meters to +76,797.65625 meters.
Altitude	An 16-bit 2's complement value for the track's height relative to the altitude of the data link reference point. The LSB represents 2.34375 meters. The altitude range is from -76,800.00000 meters to +76,797.65625 meters.
X-Velocity	An 8-bit 2's complement value for the track's rate of travel in the true East direction. The LSB represents 2.856 meters per second. The X-Velocity range is from -365.568 meters per second to +362.712 meters per second.
Y-Velocity	An 8-bit 2's complement value for the track's rate of travel in the true North direction. The LSB represents 2.856 meters per second. The Y-Velocity range is from -365.568 meters per second to +362.712 meters per second.
Time Tag	An 8-bit unsigned value representing the partial TPQ36A system clock at the time the track described in this message was scanned. The LSB represents 0.0625 seconds. The max value is 15.9375 seconds.
Checksum	An 8-bit unsigned number representing the MOD 256 sum off all the bytes in this message except Sync Byte #1, Sync Byte #2, and Checksum.

TABLE 3
ORDER OF TRANSMISSION - TRACK MESSAGE

The track message will be sent in the following order a byte at a time starting with Byte 0.

	MSB							LSB
Byte 0	Sync Byte #1							
Byte 1	Sync Byte #2							
Byte 2	Msg Type	Spare			ArType	Track ID		
Byte 3	Track Number							
Byte 4	X-Position (MS Byte)							
Byte 5	X-Position (LS Byte)							
Byte 6	Y-Position (MS Byte)							
Byte 7	Y-Position (LS Byte)							
Byte 8	Altitude (MS Byte)							
Byte 9	Altitude (LS Byte)							
Byte 10	X-Velocity							
Byte 11	Y-Velocity							
Byte 12	Time Tag							
Byte 13	Checksum							

TABLE 4
TIME MESSAGE

		RANGE	RESOLUTION
Sync Byte 1	8		
Sync Byte 2	8		
Message Type	2		
Spare	6		
Spare	8		
TPQ36A Time	32	119.3 hours	100 microsec
TOD Offset	32	119.3 hours	100 microsec
Spare	8		
Checksum	8		
	112 bits = 14 bytes		

TABLE 5
TIME MESSAGE CONTENTS DESCRIPTION

Sync Byte #1	An 8-bit number used with Sync Byte #2 by the receive processor to determine the start of the message. It has a constant value of 'AA'H.
Sync Byte #2	An 8-bit number used with Sync Byte #1 by the receive processor to determine the start of the message. It has a constant value of '55'H.
Message Type	A 2-bit word indicating the type of message. For this message, the message type = '10'B.
Spare	All spare bits and bytes shall be set to zero.
TPQ36A Time	A 32-bit unsigned value representing the system time at the time this message was sent from the radar. The LSB represents 100 microseconds. The max value is 429,496.7 seconds for approximately 119.3 hours.
TOD Offset	A 32-bit unsigned value representing the offset to the Time (above) to convert the system Time to the time-of-day. This offset is manually entered into the radar after initialization and remains constant until manually changed. The LSB represents 100 microseconds.
Checksum	An 8-bit unsigned number representing the MOD 256 sum of all the bytes in this message except Sync Byte #1, Sync Byte #2, and Checksum.

TABLE 6
TIME MESSAGE ORDER OF TRANSMISSION

The time message will be sent in the following order a byte at a time starting with Byte 0.

	MSB		LSB
Byte 0	Sync Byte #1		
Byte 1	Sync Byte #2		
Byte 2	Msg Type	Spare	
Byte 3	Spare		
Byte 4	TPQ36A Time (MS Byte)		
Byte 5	TPQ36A Time		
Byte 6	TPQ36A Time		
Byte 7	TPQ36A Time (LS Byte)		
Byte 8	TOD Offset (MS Byte)		
Byte 9	TOD Offset		
Byte 10	TOD Offset		
Byte 11	TOD Offset (LS Byte)		
Byte 12	Spare		
Byte 13	Checksum		

WEAPONS DISPLAY UNIT

Referring now to Figure 3, an illustration of the weapons display 26 of computer terminal 24 during an engagement is illustrated. Display 26, which may be reproduced on any number of visual displays but, in the preferred embodiment, is presented on a electroluminescent display integral with the computer terminal, illustrates on the situation display side 78, the targets and their relationship to the weapon location and, on the parameter display side 80, parameters associated with the selected target. As can be seen on the situation display, helicopters 82 are represented by symbols different from those utilized to represent aircraft 84, in this case a bar over the symbol indicating it is a helicopter. In this representation, a diamond indicates that the target is hostile, a U that it is unknown and a circle that the target is friendly. The azimuth line 86 indicates the current pointing direction of the weapon and the box 88 the position of a target 90 being selected or "hooked". A center vertical bar 92 contains a "bug" 94 of the desired elevation of the weapon in order to acquire the target and a traveling indicator of instantaneous elevation angle 96 of the weapon. The data 98 on the parameter display contain important information associated with the target being designated.

Referring to Figure 4, an alternate display 100 can present a menu of parameters which can allow the operator to customize the parameters displayed by his scenario and engagement data.

Figure 5 illustrates a simplified view of the visual sight 32 employed for the present invention. As shown, it is mounted close to the weapon launcher 34 and includes a boresight 104 parallel to that of the bore of the weapon launcher.

As illustrated in greater detail in Figure 6, the sight

32 employs light emitting diodes 105 in the periphery of the visual sight 32 thereby providing an indication of target cueing to the operator. In the example shown, red light emitting diodes 106 are used to indicate azimuth and elevation commands received from the computer terminal and green light emitting diodes 108 are used to provide a range indication for firing cueing. These diodes may be driven by signals from the message processing module 72. In operation, a red light emitting diode 106 will blink if it is necessary to move the angle of the launcher in that direction to acquire the target. The green light emitting diodes 108 are off until such time as the target is in range as derived from data being generated by the computer terminal, at which time they come on, indicating that the target is within range of the weapon.

Thus, the complex acquisition and firing cues normally performed visually by the weapon operator are reduced to simple steering commands until such time as the target is acquired and the very difficult "in range" estimate is reduced to a simple visual indication in the sight that the target is within range of the weapon.

It can be seen from the above description that the present invention provides for a substantially enhanced probability of acquisition and successful engagement of a target by a manpad launched system compared to the independent use of such weapons in a battlefield environment. The fact that this capability is accomplished without the necessity of proliferation of sensor sites or an increase in the vulnerability of various weapon sites is an important improvement in such air defense weapon systems. Furthermore, all of the capabilities normally associated with manpad weapon systems remain intact in the event of a failure or enemy destruction of any of the other components of the system, thereby providing with graceful degradation in the event of attack and high survivability of the individual units in the

event of a concentrated attack against any part of the system.

It will be appreciated from the above discussion that a variety of visual guidance displays may make use of the present invention. The simplified peripheral vision guidance display briefly described above, may be replaced, for instance, by virtual image displays involving the projection by holograms or otherwise of information displayed on a cathode ray tube or other visual graphics system. If such displays were used, of course, more information could be displayed than is used in the simplified peripheral vision guidance display described above and, depending upon the capabilities and training of the operator and the display capabilities of such display, it may even be possible to eliminate the requirement for a separate person to select and feed targets to the operator thereby further reducing the personnel requirement of a weapon launching location to a single operator. Furthermore, it may be seen that the invention could be applied in a similar way to air or surface borne vehicles thereby substantially reducing the acquisition time and accuracy of sensorless vehicles or vehicles with limited sensor capabilities. Thus, the application of the present invention is not limited to weapon locations with a two man crew.

As illustrated in the exemplary configuration discussed above, the invention may be configured by the use of many components that currently are available or easily modified from available equipment developed for other military purposes. Therefore, the beneficial effects of the invention do not require or rely upon the development of heretofore unavailable component technology or other technological breakthroughs but, nonetheless, provide heretofore unrealizable capabilities, efficiencies and economies in air defense weaponry. Naturally, the development or availability of special purpose components for such a system can only enhance and improve its other beneficial capabilities. While a particular form of the

invention has been illustrated and described, it will also be apparent that various modifications can be made without departing from the spirit and scope of the invention. Accordingly, it is not intended that the invention be limited except as by the appended claims.

APPENDIX

```
$NOLIST

MODULE tablemgr ;

$LARGE(subsys1 HAS pqclose; EXPORTS pqclose)
PUBLIC PQCLOSE;
PROCEDURE pqclose(VAR f:BYTES);
$COMPACT(-const in code-)

(*----------------------------------------
(*  Include files used by WDU system
(*
(*----------------------------------------*)

$INCLUDE ('w'incs'common.inc-text-)
$INCLUDE ('w'incs'conpas.inc-text-)
$INCLUDE ('w'incs'windowtypes.inc-text-)
$INCLUDE ('w'incs'windowprocs.inc-text-)
$INCLUDE ('w'incs'ospastypes.inc-text-)
$INCLUDE ('w'incs'ospasprocs.inc-text-)

(*-----------------------------------
(*  Interface section
(*
(*-----------------------------------*)


PUBLIC alert;
PROCEDURE beep;
PROCEDURE zint;
PROCEDURE zeroinit;


PUBLIC tablemgr ;
CONST       syn1         = 0AAH;
            syn2         = 55H;
            io_buff_lng  = 11;
            maxentry     = 16;
TYPE
            byte_int     =  RECORD
                              CASE BOOLEAN OF
                                TRUE : (b:ARRAY [1..2] of BYTE) ;
                                FALSE: (i:INTEGER);
                            END ; { record }
            coord_type   =  ARRAY [1..3] of LONGINT ;
                               { LSB of 2.34375  }
            cal_type     =  ARRAY [0..4,0..4] of REAL;
            clock_type   =  RECORD
                             CASE BOOLEAN OF
                               TRUE : (b:ARRAY [1..4] of BYTE) ;
                               FALSE: (i:LONGINT);
                            END ; { record }
            string8      =  packed array [1..8] of char;
            erectype     =  RECORD
```

```
                    xpos: INTEGER ;
                    ypos: INTEGER ;
                    mlng: BYTE ;
                    val : string8;
                    editt: byte;
                   END;

VAR
        erec          :  ARRAY [0..maxentry] of erectype;
        todclock      :  clock_type ;
        capture       :  BOOLEAN;
        hook_x        :  INTEGER;
        hook_y        :  INTEGER;
        hook_z        :  INTEGER;
        io_buffer     :  ARRAY [1..io_buff_lng] of BYTE ;
        in_range_bit:    INTEGER ;
        status_bit    :  INTEGER ;
        led_bit       :  INTEGER ;
        hook_a_track:    BOOLEAN ;
        hook_az       :  REAL ;
        hook_el       :  REAL ;
        hook_range    :  REAL ;
        hook_alt      :  REAL ;
        sting_az      :  byte_int ;
        sting_el      :  byte_int ;
        sting_on      :  BOOLEAN ;
        grid_mag_north:REAL ;
        dlrp_coord    :  coord_type ;        { DLRP x, y, and z coordinates
        fu_coord      :  coord_type ;
                             { Fire Unit x, y, and z coordinates  }
        alert_range   :  INTEGER  ;
        optr          :  windowregionptr ;
        opoint        :  point ;
        orect         :  rectangle ;
        fonrec        :  FONTINFORECORD ;
        lfont,bfont   :  fontpointer ;
        cal_tbl       :  cal_type ;
        el_off        :  REAL;
        mptr          :  windowregionptr ;
        dumptr        :  windowregionptr ;
        me,pe         :  WORD;
        audible       :  BOOLEAN;
        pri_num       :  INTEGER;
        hook_num      :  INTEGER;          (*rca*)
        man_hook      :  BOOLEAN;          (*rca*)
        auto_hook     :  BOOLEAN;          (*rca*)
        demo_mode     :  BOOLEAN;
        lockon        :  BOOLEAN;
        pridisp       :  BYTE ;
        startps       :  REAL ;
        endps         :  REAL ;
        pvel          :  REAL ;
        palt          :  REAL ;
```

```
PUBLIC tabl mgr ;
PROCEDURE msg_handler ;
PROCEDURE hook_process ;
PROCEDURE sting_hook;
PROCEDURE putit;
PROCEDURE beepit;

PUBLIC iointerface ;
PROCEDURE comm_init(man_dial:BOOLEAN) ;
PROCEDURE iowrapup;

PUBLIC iointerface ;
VAR    iaok: BOOLEAN;

PUBLIC stingerhandler ;
PROCEDURE sting_init ;

PUBLIC stingerhandler ;
VAR   sp : WORD;

PUBLIC parminterface ;
PROCEDURE update_parm(tf:BOOLEAN) ;

PUBLIC builtinradar ;
PROCEDURE activateradar ;
PROCEDURE deactivateradar ;

(*-----------------------------------------
(*      GRID WDU software
(*
(*-----------------------------------------*)
$NOLIST
PROGRAM tablemgr ( calfil, capfil, input , output ) ;
CONST
{-----------------------------------------
{     CONSTANTS used in GRID interface system
{
{-----------------------------------------}
time_lsb     = 0.0625  ;          { TPQ time units }
pos_lsb      = 2.34375 ;          { TPQ coordinate units }
pos_lsb_sq   = 5.4921640625 ;     { TPQ coordinate units squared }
vel_lsb      = 2.856   ;          { TPQ velocity units }
time_vel_lsb = 0.1785  ;          { time units * velocity units }
del_rate     = 72; (*112-oserial*) { 12.0 secs / .0625 =
                                        delete rate }
az_lsb       = 0.125   ;          { azimuth lsb 1/8 degree }
max_track    = 12      ;          { track table size}
hi_velocity  = 200.0   ;          { velocity boundary }
lo_velocity  = 50.0    ;          { velocity boundary          }
time_msg     = 2       ;          { TPQ time msg               }
track_msg    = 1       ;          { TPQ track msg              }
cr           = 13      ;
deg_rad      = 1.7453292520E-02 ;
deg_rad_az   = 2.1816615650E-03 ;
```

```
rad_deg      = 5.7295779513E+1 ;
deg_mil      = 17.77777 ;
win_dim      = 250 ;
time_delay   = 60 ;              (* delay timer before bump clock 1/16 sec *)
time_pri     = 0 ;
sound_pri    = 10;
cap_pri      = 10;
prog_pri     = 210 ;
sting_min = 1000.0;
sting_max    = 5000.0;
zero         = 222;
step         = 0.0981747704;
capname      = ' 'w'dat'data~text~';
capmax       = 400000;

TYPE
{-------------------------------------------
{      TYPES    used in GRID interface system
{
{-------------------------------------------}
state_type    = (idle,ball_pend,ball_active,num_pend);
char_byte     =  RECORD
                   CASE BOOLEAN OF
                     TRUE : (c:CHAR) ;
                     FALSE: (b:BYTE) ;
                  END ; { record }
track_type    =  RECORD
            active    :  BOOLEAN ;   { active-true , inactive-false }
            track_id  :  INTEGER ;   { friend,hostile,unknown,test  }
            track_fix:   BOOLEAN ;
            track_num:   INTEGER ;
            x_pos     :  byte_int;   { LSB of 2.34375 meters        }
            y_pos     :  byte_int;   { LSB of 2.34375 meters        }
            altitude  :  byte_int;   { LSB of 2.34375 meters        }
            x_vel     :  INTEGER ;   { LSB of 2.856 meters / sec    }
            y_vel     :  INTEGER ;   { LSB of 2.856 meters / sec    }
            dx_vel    :  INTEGER ;   { LSB of 2.856 meters / sec    }
            dy_vel    :  INTEGER ;   { LSB of 2.856 meters / sec    }
            abell     :  BYTE    ;   (*30aug85*)
            time_tag  :  INTEGER ;   { LSB of .0625 seconds         }
           first_site: LONGINT ;
           track_weap: BYTE    ;   (* 21MAR86 *)
                END ; { record }
                  caprec        =  RECORD
                  cap_time :  LONGINT ; (*4*)
                  cap_num  :  BYTE ;    (*1*)
                  cap_vel  :  REAL ;    (*4*)
                  cap_range:  REAL ;    (*4*)
                  cap_az   :  REAL ;    (*4*)
                  cap_alt  :  REAL ;    (*4*)
                  cap_prio :  REAL ;    (*4*)
                END ;                   (*25*)
screen_type   =  RECORD
```

```
                screen_num    : INTEGER ;  ( screen track number)
                screen_x      : INTEGER ;  ( screen x coord      )
                screen_y      : INTEGER ;  ( screen y coord      )
                screen_id     : INTEGER ;  ( screen id number    )
                screen_vel    : REAL    ;  ( screen y coord      )
                screen_el     : REAL    ;  ( screen y coord      )
                screen_fix    : BOOLEAN ;  ( screen y coord      )
                screen_head   : REAL    ;  ( screen y coord      )
                screen_dhead  : REAL    ;
                screen_az     : REAL    ;
                screen_2drange: REAL    ;
                screen_3drange: REAL    ;
                screen_alt    : REAL    ;
                screen_pcp    : REAL    ;
                screen_prio   : REAL    ;   (*30aug85*)
                screen_weap   : BYTE    ;   (*21MAR86*)
              END ;
rank_rec    = RECORD
                rank_num : INTEGER ;
                rank_prio: REAL    ;
              END ;
weap_type   = PACKED ARRAY [1..6] of CHAR ;

VAR
{-----------------------------------------------
{      VARIABLES used in GRID interface system
{
{-----------------------------------------------}
capfil        : FILE of caprec;
capful        : LONGINT;
testfn        : PACKED ARRAY [1..18] of CHAR;
mils          : INTEGER;
msg_there     : BOOLEAN ;
stop          : BOOLEAN ;
mrect         : rectangle ;
mpoint        : point ;
mmode         : WORD ;
x,y,z         : INTEGER ;
toffset       : clock_type ;
tclock        : clock_type ;
clock         : clock_type ;       { GRID real time clock              }
track_rec     : track_type ;       { hyolds one record from track table }
track_tbl     : ARRAY [0..max_track] of track_type ;
screen_tbl    : ARRAY [0..max_track] of screen_type ;
rank_tbl      : ARRAY [0..max_track] of rank_rec ;
stemp         : INTEGER ;
hook_idx      : INTEGER ;
temp          : INTEGER  ;
locked_on     : BOOLEAN  ;
meters_pixel  : INTEGER ;          { screen dimension*1000 / 250   }
screen_fu_x   : INTEGER ;          { screen fu x coord default 125 }
screen_fu_y   : INTEGER ;          { screen fu y coord default 125 }
hook_id       : INTEGER ;          { hokked id type                }
```

```
hook_vel      :  REAL    ;          { hooked velocity                }
hook_head     :  REAL    ;          { hooked heading                 }
ho k_ttc      :  REAL    ;          { hooked time to close           }
hook_pcp      :  REAL    ;          (* predicted closest point *)
hook_dhead    :  REAL    ;          (* to maneaver or not to maneuver *)
hook_rank     :  INTEGER ;
hook_weap     :  BYTE    ;
tsound_alert  :  BOOLEAN ;          (*30aug85*)
in_process    :  BOOLEAN ;
t_hook_num    :  INTEGER ;
ccount        :  INTEGER ;
kbuff         :  ARRAY [1..3] of BYTE ;
state         :  state_type ;
ball_x        :  INTEGER ;
ball_y        :  INTEGER ;
calfil        :  TEXT ;
errorw        :  WORD ;
stime         :  timetype;
temp_al       :  INTEGER ;
cap_pid,csid  :  WORD ;
time_pid      :  WORD ;
sting_pid     :  WORD ;
prog_pid      :  WORD ;
io_pid        :  WORD ;
fonconn       :  WORD ;
fonres        :  BYTE ;
fonpath       :  PACKED ARRAY [1..27] of CHAR ;
error         :  WORD ;
error1        :  WORD ;
error2        :  WORD ;
adjust        :  INTEGER ;
mph           :  BOOLEAN ;
fake_radar    :  BOOLEAN ;
elevptr       :  windowregionptr;  (*1/14/86*)
elevrect      :  rectangle;        (*1/14/86*)
elevpoint     :  point;            (*1/14/86*)
t_inrange     :  clock_type;
t_lockon      :  clock_type;
t_hooktim     :  clock_type;
t_hooknum     :  INTEGER;
rank          :  BOOLEAN;
weap_buff     :  ARRAY [0..3] of weap_type;
memque        :  ARRAY [0..255] of caprec;  (*28apr86*)
mempos        :  BYTE;                      (*28apr86*)


(*-----------------------------
(*  display the T.O.D.
(*-----------------------------*)
PROCEDURE showtime(t:clock_type);
var lint     : LONGINT ;
    junk     : clock_type ;
BEGIN
```

```
lint := t.i DIV 10000 ;
junk.i := t.i DIV 625 ;
WRITE((((lint) DIV 3600) MOD 3600):2,':',
      (((lint) DIV 60) MOD 60):2,':',
      ((lint) MOD 60):2,':',
      TRUNC((junk.b[1] MOD 16)*0.625):1);
END;

(*-----------------------------
(* common screen i/o to save room
(*-----------------------------*)
PROCEDURE writexyr(x,y:INTEGER;r:REAL;l,d:INTEGER);
BEGIN
CONMOVECSR(x,y);
WRITE(r:l:d);
END;

(*-----------------------------
(* evaluate performance screen
(*-----------------------------*)
PROCEDURE evaluate;
VAR ch:CHAR;x:INTEGER;t_first_site:clock_type;
BEGIN
WINCOPYREMOTERECTANGLE(NIL,optr,orect,opoint,0);
CONRESETDISPLAY;
CONDEFCSR(FALSE);
CONMOVECSR(2,2);
CONLINEOUT('TRACK NUMBER   :',16);
CONMOVECSR(2,4);
CONLINEOUT('FIRST SITE TIME:',16);
CONMOVECSR(2,6);
CONLINEOUT('HOOKED TIME    :',16);
CONMOVECSR(2,8);
CONLINEOUT('ON TARGET TIME :',16);
CONMOVECSR(2,10);
CONLINEOUT('IN RANGE TIME  :',16);
CONMOVECSR(2,12);
CONLINEOUT('** <esc> to return to monitor **',32);
x:=1;
WHILE (x<=max_track) AND (t_hooknum<>track_tbl[x].track_num) DO x:=x+1;
IF (x<=max_track) THEN t_first_site.i:=track_tbl[x].first_site ELSE
t_first_site.i:=0;
CONMOVECSR(20,2);
WRITE(t_hooknum:10);
CONMOVECSR(20,4);
showtime(t_first_site);
CONMOVECSR(20,6);
showtime(t_hooktim);
CONMOVECSR(20,8);
showtime(t_lockon);
CONMOVECSR(20,10);
showtime(t_inrange);
REPEAT
```

```
 ch:=CONCHARIN; UNTIL ch=CHR(27);
WINCOPYREMOTERECTANGLE(optr,NIL,orect,opoint,0);
CONDEFCSR(FALSE);
END;


(*----------------------------
(* load calibration table
(*--------------------------*)
PROCEDURE calib_data;
VAR conn,error,error1:WORD;
    fname:PACKED ARRAY [1..22] of CHAR;
    x,y:INTEGER;

BEGIN
CONLINEOUT(' V2.0 Loading Calibration Data.....',35);
error1:=0;
fname:=' 'b'dat'gridcalb~text~';
fname[1]:=CHR(21);
conn := OSATTACH(fname,1,error1,3,error);
OSDETACH(conn,error1);
CONHEXOUT(error);
FOR x := 0 to 4 DO
  FOR y := 0 to 4 DO
    cal_tbl[x,y]:=0.0;
el_off:=0.0;
IF error=0 THEN
  BEGIN
  RESET(calfil,''b'dat'gridcalb~text~');
  FOR x := 0 to 4 DO
    FOR y := 0 to 4 DO
      READLN(calfil,cal_tbl[y,x]);
  READLN(calfil,el_off);
  PQCLOSE(calfil);
  END;
END;

(*----------------------------
(* 1/16 second timer
(*--------------------------*)
PROCEDURE timer ;
BEGIN
REPEAT
  OSDELAY(time_delay);
  adjust := ( adjust + 1 ) MOD 3 ;
  IF ( adjust = 0 ) THEN
    BEGIN
    clock.i := clock.i + 2 ;
    todclock.i := todclock.i + 1250 ;
    END
   ELSE
    BEGIN
```

```
    clock.i := clock.i + 1 ;
    todclock.i := todclock.i + 625 ;
    END ;
UNTIL FALSE ;
END;

(*----------------------------
(* quick circle
(*----------------------------*)
PROCEDURE wincircle(c1,c2,r:INTEGER);
VAR s:real;xo,yo,x:INTEGER;
BEGIN
s:=0;
FOR x:=1 to 8 DO
  BEGIN
  xo:=TRUNC(r*SIN(s));
  yo:=TRUNC(r*COS(s));
  WINDRAWPIXEL(c1+xo,c2+yo);
  WINDRAWPIXEL(c1+xo,c2-yo);
  WINDRAWPIXEL(c1-xo,c2+yo);
  WINDRAWPIXEL(c1-xo,c2-yo);
  WINDRAWPIXEL(c1+yo,c2+xo);
  WINDRAWPIXEL(c1+yo,c2-xo);
  WINDRAWPIXEL(c1-yo,c2+xo);
  WINDRAWPIXEL(c1-yo,c2-xo);
  s:=s+step;
  END;
END;

(*----------------------------
(* data capture  task
(*----------------------------*)
PROCEDURE putit;
VAR e1,e2:WORD;
BEGIN
REPEAT
  e2:=OSWAIT(csid,0ffffh,e1);
  IF capture THEN
    BEGIN
    capfil^:=memque[e2];  (*28apr86*)
    PUT(capfil);
    END;
UNTIL FALSE;
END;

(*----------------------------
(* data capture  routine
(*----------------------------*)
PROCEDURE captureit(trck:screen_type);
VAR e1,e2,errorw:WORD;
BEGIN
(*WITH capfil^ DO*)     (*28apr86*)
WITH memque[mempos] DO (*28apr86*)
```

```
  BEGIN
  WITH trck DO
    BEGIN
    cap_time:= todclock.i;
    cap_num := screen_num;
    cap_vel := screen_vel;
    cap_range:=screen_3drange;
    cap_az  := screen_az;
    cap_alt := screen_alt;
    cap_prio:= screen_prio;
    END;
  END;
el:=mempos;              (*28apr86*)
OSSIGNAL(csid,1,el,e2);
mempos:=mempos+1;        (*28apr86*)
capful:=capful+1;
IF capful>capmax THEN
  BEGIN
  capture:=FALSE;
  capful:=0;
  CONMOVECSR(30,0);
  CONLINEOUT('DSKFUL',6);
  PQCLOSE(capfil);
  END;
END;

(*--------------------------------
  Procedure msg-handler
   PURPOSE : converts io_buffer ( array of bytes ) to record format
             locates position in track_table for track_rec
             r clock
   INPUT   : io_buffer
             io_start
   OUTPUT  : clock
             track_tbl
 --------------------------------*)
PROCEDURE msg_handler ;
VAR count     : INTEGER ;
    last      : INTEGER ;
    tmsg_type: INTEGER ;
    ttrack_rec:track_type ;
    tbyte     : BYTE ;

BEGIN
tmsg_type := io_buffer[1] DIV 64 ;
IF tmsg_type = time_msg THEN
  BEGIN
  tclock.b[4] := io_buffer[3] ;
  tclock.b[3] := io_buffer[4] ;
  tclock.b[2] := io_buffer[5] ;
  tclock.b[1] := io_buffer[6] ;
  clock.i := (tclock.i DIV 625) + 7;      (* add 7/16 sec to not drop *)
  toffset.b[4] := io_buffer[7] ;
```

```
toffset.b[3] := io_buffer[8] ;
toffset.b[2] := io_buffer[9] ;
toffset.b[1] := io_buffer[10] ;
todclock.i := tclock.i + toffset.i ;
END
ELSE
IF tmsg_type = track_msg THEN
  BEGIN
  WITH ttrack_rec DO
    BEGIN
    ttrack_rec.abell := 0 ;
    ttrack_rec.dx_vel:= 0 ;
    ttrack_rec.dy_vel:= 0 ;
    active := TRUE ;
    ttrack_rec.first_site := todclock.i;
    track_id := io_buffer[1] MOD 4 ;
    track_fix := ((io_buffer[1] DIV 4) MOD 2) = 1;
    track_weap:= ((io_buffer[1] DIV 8) MOD 4);
    track_num := io_buffer[2] ;
    x_pos.b[2] := io_buffer[3] ;
    x_pos.b[1] := io_buffer[4] ;
    y_pos.b[2] := io_buffer[5] ;
    y_pos.b[1] := io_buffer[6] ;
    altitude.b[2] := io_buffer[7] ;
    altitude.b[1] := io_buffer[8] ;
    x_vel := io_buffer[9] ;
    IF x_vel >= 128 THEN x_vel := x_vel-256 ;
    y_vel := io_buffer[10] ;
    IF y_vel >= 128 THEN y_vel := y_vel-256 ;
    time_tag := io_buffer[11] ;
    {------ insert in table ----------}
    last := 0 ;
    count := 1 ;
    WHILE(ttrack_rec.track_num<>track_tbl[count].track_num)AND
         (count<=max_track)DO
      BEGIN
      IF NOT track_tbl[count].active THEN last := count ;
      count := count + 1 ;
      END ;
    IF count <=  max_track THEN
      BEGIN
      ttrack_rec.dx_vel:=(ttrack_rec.x_vel-track_tbl[count].x_vel);
      ttrack_rec.dy_vel:=(ttrack_rec.y_vel-track_tbl[count].y_vel);
      ttrack_rec.abell := track_tbl[count].abell;
      IF track_tbl[count].first_site<>0 THEN
        ttrack_rec.first_site := track_tbl[count].first_site;
      track_tbl[count] := ttrack_rec ;
      END
     ELSE
      IF last <> 0 THEN track_tbl[last] := ttrack_rec ;
    IF (track_id=1) AND auto_hook AND (NOT man_hook) THEN  (*rca*)
      BEGIN                                                (*rca*)
      hook_num:=track_num;                                 (*rca*)
```

```
        hook_a_track:=TRUE;                                    (*rca*)
        END;
      END ; { with ttrack_rec }
    END ; { if .. then }
END ; { procedure msg_handler }


{-------- initialize screen and variables --------}
PROCEDURE initialize ;
VAR count ,x,y,page,line : INTEGER ;
BEGIN
zeroinit;
weap_buff[0]:='  NONE';
weap_buff[1]:=' TIGHT';
weap_buff[2]:='  HOLD';
weap_buff[3]:='  FREE';
t_hooktim.i:=0;
t_inrange.i:=0;
t_lockon.i:=0;
t_hooknum:=0;
iaok:=FALSE;
mph:=FALSE;
lockon:=FALSE;
fake_radar:=FALSE;
man_hook:=FALSE;
hook_az:=0;
hook_el:=0;
hook_range:=0;
hook_vel:=0;
hook_ttc:=0;
hook_weap:=0;
capture:=FALSE;
mils:=1;
me:=0;
pe:=0;
CONRESETDISPLAY;
calib_data;
fonres := 0 ;
fonpath := ' ''programs'ascii9x12~font~';
fonpath[1] := CHR(26);
fonconn := OSATTACH(fonpath,1,fonres,1,error);
OSOPEN(fonconn,1,error);
bfont := WINLOADFONT(fonconn,error);
lfont := WINSETFONT(bfont,fonrec,0);
OSDETACH(fonconn,error);
OSCLOSE(fonconn,error);
adjust := 0 ;
clock.i := 0 ;
todclock.i := 0 ;
sting_on := FALSE ;
csid:=OSCREATESEMAPHORE(errorw);
cap_pid:=OSFORKPROCESS(putit,10,FALSE,500,errorw);
```

```
time_pid := OSFORKPROCESS(timer,time_pri,FALSE,500,errorw);
IF errorw <> 0 THEN WRITELN('fork error- ',errorw);
prog_pid := OSWHOAMI ;
OSSETPRIORITY(prog_pid,prog_pri,errorw);
IF errorw <> 0 THEN WRITELN('set error- ',errorw);
mpoint.x:= 1;
mpoint.y:= 1;
mrect.topleft.x := 1 ;
mrect.topleft.y := 1 ;
mrect.extent.x := win_dim ;
mrect.extent.y := win_dim ;
opoint.x:= 0;
opoint.y:= 0;
orect.topleft.x := 0 ;
orect.topleft.y := 0 ;
orect.extent.x := 512 ;
orect.extent.y := 256 ;
state := idle ;
stop := FALSE ;
hook_idx := 0 ;
hook_a_track := FALSE ;
hook_num := 0 ;
screen_fu_x := win_dim DIV 2 ;
screen_fu_y := win_dim DIV 2 ;
ball_x:=screen_fu_x;
ball_y:=screen_fu_y;
meters_pixel := 80 ;    (* 160km / 250 *)
FOR count := 1 to max_track DO track_tbl[count].active := false ;
CONRESETDISPLAY;.mptr :=
WINALLOCATEWINDOWMEMORY(win_dim,win_dim,screenformat,errorw);
optr := WINALLOCATEWINDOWMEMORY(512,256,screenformat,errorw);
dumptr := WINALLOCATEWINDOWMEMORY(1,1,screenformat,errorw);
elevptr:= WINALLOCATEWINDOWMEMORY(13,256,screenformat,errorw); (*1/14/86*)
FOR count := 0 to maxentry DO erec[count].val := '0       ' ;
erec[7].val:='15000   ';
update_parm (TRUE);
sting_az.i := 0 ;
sting_el.i := 0 ;
CONMOVECSR(30,2);CONLINEOUT('TIME TO ENG  :       sec',25);
CONMOVECSR(30,3);CONLINEOUT('RANGE        :       km ',25);
CONMOVECSR(30,4);CONLINEOUT('AZIMUTH      :       mil',25);
CONMOVECSR(30,5);CONLINEOUT('ELEVATION    :       mil',25);
CONMOVECSR(30,6);CONLINEOUT('ALTITUDE     :       m  ',25);
CONMOVECSR(30,7);CONLINEOUT('VELOCITY     :       m/s',25);
CONMOVECSR(30,8);CONLINEOUT('TRACK NUMBER :',14);
CONMOVECSR(30,9);CONLINEOUT('PRIORITY     :',14);
(*
CONMOVECSR(30,10);CONLINEOUT('WEAPON CNTRL :',14);
*)
WINDRAWLINE(0,0,win_dim+1,0);
WINDRAWLINE(win_dim+1,0,win_dim+1,win_dim+1);
WINDRAWLINE(win_dim+1,win_dim+1,0,win_dim+1);
WINDRAWLINE(0,win_dim+1,0,0);
```

```
WINDRAWLINE(264,0,510,0);          (*1/14/86*)
WINDRAWLINE(510,0,510,251);
WINDRAWLINE(510,251,250,251);      (*1/14/86*)
WINDRAWLINE(264,251,264,0);        (*1/14/86*)
CONMOVECSR(48,15);CONLINEOUT('HUGHES',6);
FOR x := 205 to 227 DO       (*425,205--491.227*)
  WININVERTLINE(425,x,491,x);
WINERASELINE(425,205,426,205);
WINERASELINE(491,205,490,205);
WINERASELINE(425,227,426,227);
WINERASELINE(491,227,490,227);
WINERASELINE(425,227,425,226);
WINERASELINE(491,227,491,226);
WINERASELINE(425,205,425,206);
WINERASELINE(491,205,491,206);
msg_there:=FALSE;
if NOT demo_mode THEN comm_init(FALSE);
sting_init;
END ; { proc }

(*-----------------------------
(* compute azimuth
(*---------------------------*)
FUNCTION pazimuth(y,x:real):REAL;
BEGIN
IF (x=0.0) THEN x:=0.1;
IF x>=0 THEN
  pazimuth:=90.0-(ARCTAN(y/x)*rad_deg)
 ELSE
  pazimuth:=270.0-(ARCTAN(y/x)*rad_deg);
END;

(*-----------------------------
(* compute heading
(*---------------------------*)
FUNCTION pheading(y,x:INTEGER):REAL;
BEGIN
IF (x=0) THEN x:=1;
IF x>=0 THEN
  pheading:=90.0-(ARCTAN(y/x)*rad_deg)
 ELSE
  pheading:=270.0-(ARCTAN(y/x)*rad_deg);
END;

(*-----------------------------------*)
(* PCP computes and displays         *)
(*     predicted closest   point     *)
(*-----------------------------------*)
FUNCTION pcp(range,az,head:real):REAL;
VAR beta:REAL;
BEGIN
beta:=abs(abs(az-head)-180.0);
IF (beta<90) THEN
```

```
  pcp:=(range*(cos((90-beta)*deg_rad)))
 ELSE
  pcp:=range;
END;


{----------------------------------------
{  Procedure map_data
{   PURPOSE : computes screen pixel location
{             computes velocity
{   INPUT   : track_rec ( TPQ data )
{             time_clock
{             meters_pixel
{             pixel_fu_x
{             pixel_fu_y
{   OUTPUT  : screen_num
{             screen_id
{             screen_x
{             screen_y
{             screen_vel
{----------------------------------------}
PROCEDURE map_data(idx,idx2:INTEGER);
VAR time_laps : BYTE ;
    x_rel_fu  : real;
    y_rel_fu  : real;
BEGIN
WITH track_rec DO
  BEGIN
  WITH screen_tbl[idx] DO
    BEGIN
    screen_num := track_num ;
    screen_fix := track_fix ;
    screen_id := track_id ;
    screen_weap := track_weap ;
    screen_vel := (SQRT(x_vel*x_vel+y_vel*y_vel)*vel_lsb) ;
    time_laps := ( clock.b[1] )-time_tag;
    x_rel_fu:=((pos_lsb*x_pos.i)+(x_vel*time_laps*time_vel_lsb)
              -fu_coord[1]);
    y_rel_fu:=((pos_lsb*y_pos.i)+(y_vel*time_laps*time_vel_lsb)
              -fu_coord[2]);
    screen_x := screen_fu_x+TRUNC(x_rel_fu/meters_pixel);
    screen_y := screen_fu_y-TRUNC(y_rel_fu/meters_pixel);
    screen_head:=pheading(y_vel,x_vel);
    screen_az:=pazimuth(y_rel_fu,x_rel_fu);
    screen_dhead:=pheading(y_vel+dy_vel,x_vel+dx_vel);
    screen_dhead:=ABS(screen_dhead-screen_head);
    IF screen_dhead > 180.0 THEN screen_dhead:=360.0-screen_dhead;
    screen_2drange:=(SQRT(x_rel_fu*x_rel_fu + y_rel_fu*y_rel_fu));
    screen_alt:=altitude.i*pos_lsb;
    screen_3drange:=(SQRT(screen_2drange*screen_2drange+screen_
                    alt*screen_alt));
    screen_el:=(ARCTAN((screen_alt-fu_coord[3])/screen_2drange)*rad_deg) ;
    screen_pcp:=pcp(screen_3drange,screen_az,screen_head);
    IF (screen_2drange <= alert_range) THEN
      BEGIN
      IF (abell <> 2) THEN
```

```
        BEGIN
        track_tbl[idx2].abell := 1;
        tsound_alert := true;
        END
      END
     ELSE
      track_tbl[idx2].abell := 0;
    END ;
  END ;  { with track_rec }
END ; { procedure map_data }

(*-----------------------------------------*)
(*1/14/86                                   *)
(*   elevbar : represents track elevation   *)
(*             and stinger elevation via    *)
(*             bar graph                    *)
(*-----------------------------------------*)
PROCEDURE elevbar;
VAR se,te,x:INTEGER;
BEGIN
IF hook_a_track THEN
  te:=TRUNC(zero-(hook_el*2.78))
 ELSE
  te:=zero;
IF sting_on THEN
  se:=TRUNC(zero-(sting_el.i*0.3473))
 ELSE
  se:=zero;
WINSETALTERNATEWINDOW(elevptr);
WINERASEWINDOW;
WINDRAWLINE(1,zero,12,zero);
FOR x:=5 to 8 DO
  WINDRAWLINE(x,te,x,zero);
IF sting_on THEN
  FOR x:=se-1 to se+1 DO
    WININVERTLINE(1,x,12,x);
elevpoint.x:= 252;
elevpoint.y:= 1;
elevrect.topleft.x := 1 ;
elevrect.topleft.y := 1 ;
elevrect.extent.x := 12;
elevrect.extent.y := 250;
WINCOPYREMOTERECTANGLE(elevptr,NIL,elevrect,elevpoint,0);
WINSETALTERNATEWINDOW(NIL);
END;


(*----------------------------
(* display T.O.D.
(*---------------------------*)
PROCEDURE timeofday;
BEGIN
CONMOVECSR(44,11);
```

```
showtime(todclock);
END;


(*----------------------------
(* display screen dimensions
(*----------------------------*)
PROCEDURE scrndim;
VAR lint:LONGINT;
BEGIN
lint := meters_pixel DIV 4 ;
CONMOVECSR(30,11);
WRITELN(lint:3,' X ',lint:3,'KM');
END;

(*----------------------------
(* display hooked track data
(*----------------------------*)
PROCEDURE metricdata;
VAR count:INTEGER;
BEGIN
(*
CONMOVECSR(44,10);
CONLINEOUT(weap_buff[hook_weap],6);
*)
CONMOVECSR(44,8);
WRITE(hook_num:6);
CONMOVECSR(44,9);
IF (pridisp <> 0) THEN WRITE(hook_rank:6) ELSE
CONLINEOUT('NO-PRI',6);
IF NOT mph THEN writexyr(44,7,hook_vel,6,1) ELSE
writexyr(44,7,(hook_vel*3600/1852),6,1);
CASE mils OF
    0:writexyr(44,4,hook_az,6,1);
    1:writexyr(44,4,(hook_az*deg_mil),6,1);
    END; (* case *)
CASE mils OF
    0:writexyr(44,5,(hook_el),6,1);
    1:writexyr(44,5,(hook_el*deg_mil),6,1);
    END;
writexyr(44,3,(hook_range/1000),6,1);
writexyr(44,6,(hook_alt),6,1);
CONMOVECSR(44,2);
IF (hook_ttc) = -2.0 THEN CONLINEOUT('NO-ENG',6) ELSE
IF (hook_ttc) = -1.0 THEN CONLINEOUT('IN-RNG',6) ELSE
writexyr(44,2,hook_ttc,6,1);
END;

(*----------------------------
(* display w/i 4 degree window
(*----------------------------*)
PROCEDURE onthemoney;
BEGIN
```

```
IF sting_on THEN
  BEGIN
  CONMOVECSR(44,1);
  IF lock n THEN
    CONLINEOUT('ON-TARGET',9)
   ELSE
    CONLINEOUT('          ',9);
  IF (lockon) AND (t_lockon.i=0) THEN t_lockon.i:=todclock.i;
  END;
END ; ( if .. then )

(*----------------------------
(* rank the tracks per priority
(*--------------------------*)
PROCEDURE rank_priority;
VAR x,y:INTEGER;temp:rank_rec;
BEGIN
FOR x := 1 to stemp-1 DO
  FOR y :=  x+1 to stemp DO
    IF rank_tbl[y].rank_prio < rank_tbl[x].rank_prio THEN
      BEGIN
      temp:=rank_tbl[y];
      rank_tbl[y]:=rank_tbl[x];
      rank_tbl[x]:=temp;
      END;
END;

(*----------------------------
(* inside or outside priority sector
(*--------------------------*)
FUNCTION outsidesector(a:real):boolean;
BEGIN
IF endps > startps THEN
  outsidesector := NOT ((a>=startps)and(a<=endps))
 ELSE
  outsidesector := NOT ((a>=startps)or(a<=endps));
END;


(*----------------------------
(* compute priority value
(*--------------------------*)
PROCEDURE prioritize;
VAR count : INTEGER;
    tpri  : REAL;
    n     : REAL;
BEGIN
tpri:=999;
FOR count := 1 to stemp do
  WITH screen_tbl[count] DO
    BEGIN
    screen_prio:=0;
```

```
    n := 0;
    IF screen_3drange > alert_range THEN screen_prio:=512 ELSE
    IF screen_id = 2 THEN screen_prio:=256 ELSE
    IF outsidesector(screen_az) THEN screen_prio:=128 ELSE
    IF (screen_vel>=30) AND (not (screen_dhead>=2.25)) THEN
      BEGIN
      IF screen_3drange = screen_pcp THEN screen_prio := 64 ELSE
      IF screen_pcp>5000 THEN screen_prio:=32 ELSE
      IF screen_pcp<1000 THEN screen_prio:=16;
      END;
    IF screen_prio=0 THEN
      BEGIN
      IF screen_3drange > 5000 then screen_prio:=8 ELSE
      IF screen_alt > palt THEN screen_prio:=4 ELSE
      IF screen_vel > pvel THEN screen_prio:=2 ELSE
      IF (not (screen_dhead>=2.25)) THEN screen_prio:=1;
      END;
    IF screen_vel=0 THEN n:=screen_3drange/100000 ELSE
    n:=(screen_3drange/screen_vel)/100000;
    screen_prio:=screen_prio+n;
    rank_tbl[count].rank_num:=screen_num;
    rank_tbl[count].rank_prio:=screen_prio;
    IF screen_prio < tpri THEN
      BEGIN
      tpri:=screen_prio;
      pri_num:=screen_num;
      END;
    END;
END;

(*-----------------------------
(* draw priority sector
(*---------------------------*)
PROCEDURE prioritysector;
BEGIN
WINDRAWLINE(screen_fu_x,screen_fu_y,
            screen_fu_x+TRUNC(300*SIN(startps*deg_rad)),
            screen_fu_y-TRUNC(300*COS(startps*deg_rad)));
WINDRAWLINE(screen_fu_x,screen_fu_y,
            screen_fu_x+TRUNC(300*SIN(endps*deg_rad)),
            screen_fu_y-TRUNC(300*COS(endps*deg_rad)));
END;

{--------- display tracks on screen ---------}


PROCEDURE display ;
VAR r,count,x: INTEGER ;
BEGIN
WINSETALTERNATEWINDOW(mptr);
WINERASEWINDOW;
WINDRAWLINE(screen_fu_x,screen_fu_y-2,screen_fu_x,screen_fu_y+2) ;
WINDRAWLINE(screen_fu_x-2,screen_fu_y,screen_fu_x+2,screen_fu_y) ;
```

```
WINDRAWLINE(ball_x+7,ball_y-7,ball_x+7,ball_y+7);
WINDRAWLINE(ball_x+7,ball_y+7,ball_x-7,ball_y+7);
WINDRAWLINE(ball_x-7,ball_y+7,ball_x-7,ball_y-7);
WINDRAWLINE(ball_x-7,ball_y-7,ball_x+7,ball_y-7);
FOR count := 1 to stemp DO
  BEGIN
  WITH screen_tbl[count] DO
    BEGIN
     CASE screen_id OF
       0:BEGIN
         WINDRAWLINE(screen_x+3,screen_y-3,screen_x+3,screen_y+2);
         WINDRAWLINE(screen_x+2,screen_y+3,screen_x-2,screen_y+3);
         WINDRAWLINE(screen_x-3,screen_y+2,screen_x-3,screen_y-3);
         END;
       1:BEGIN
         WINDRAWLINE(screen_x+4,screen_y,screen_x,screen_y+4);
         WINDRAWLINE(screen_x-1,screen_y+3,screen_x-4,screen_y);
         END;
       2:BEGIN
         WINDRAWLINE(screen_x-1,screen_y-4,screen_x+1,screen_y-4);
         WINDRAWLINE(screen_x+2,screen_y-3,screen_x+4,screen_y-1);
         WINDRAWLINE(screen_x+4,screen_y,screen_x+4,screen_y+1);
         WINDRAWLINE(screen_x+3,screen_y+2,screen_x+1,screen_y+4);
         WINDRAWLINE(screen_x,screen_y+4,screen_x-1,screen_y+4);
         WINDRAWLINE(screen_x-2,screen_y+3,screen_x-4,screen_y+1);
         WINDRAWLINE(screen_x-4,screen_y,screen_x-4,screen_y-1);
         WINDRAWLINE(screen_x-3,screen_y-2,screen_x-2,screen_y-3);
         END;
       3:BEGIN
         WINDRAWLINE(screen_x,screen_y-4,screen_x+4,screen_y);
         WINDRAWLINE(screen_x+3,screen_y+1,screen_x,screen_y+4);
         WINDRAWLINE(screen_x-1,screen_y+3,screen_x-4,screen_y);
         WINDRAWLINE(screen_x-3,screen_y-1,screen_x-1,screen_y-3);
         END;
       END; (* case *)
    IF screen_vel > hi_velocity THEN r := 15 ELSE
    IF screen_vel > lo_velocity THEN r := 5 ELSE
    r := 0 ;
    IF r <> 0 then
      WINDRAWLINE(screen_x,screen_y,
                  screen_x+TRUNC(r*SIN(screen_head*deg_rad)),
                  screen_y-TRUNC(r*COS(screen_head*deg_rad)));
    IF screen_fix THEN WINDRAWLINE(screen_x-5,screen_y-5,screen_x+5,
                  screen_y-5);
    IF (hook_a_track) AND (hook_num = screen_num) THEN
        BEGIN
        WINDRAWLINE(screen_x+7,screen_y-7,screen_x+7,screen_y+7);
        WINDRAWLINE(screen_x+7,screen_y+7,screen_x-7,screen_y+7);
        WINDRAWLINE(screen_x-7,screen_y+7,screen_x-7,screen_y-7);
        WINDRAWLINE(screen_x-7,screen_y-7,screen_x+7,screen_y-7);
        END
       ELSE
        IF (pridisp<>0) AND (pri_num = screen_num) THEN
          WINCIRCLE(screen_x,scr en_y,8);,
```

```
    END ; { with screen_tbl }
 END ; { for .. next }
 IF sting_on THEN
   WINDRAWLINE(screen_fu_x,screen_fu_y,
               screen_fu_x+TRUNC(100*SIN(sting_az.i*deg_rad_az)),
               screen_fu_y-TRUNC(100*COS(sting_az.i*deg_rad_az)));
IF (pridisp<>0) THEN prioritysector;
WINCIRCLE(screen_fu_x,screen_fu_y,alert_range DIV meters_pixel);
WINCOPYREMOTERECTANGLE(mptr,nil,mrect,mpoint,mmode);
WINSETALTERNATEWINDOW(NIL);
IF hook_a_track THEN
  BEGIN
  metricdata;
  onthemoney;
  END;
scrndim;
timeofday;
elevbar;
END; { proc }

(*-------------------------------------*)
(* PIP computes and displays           *)
(*     predicted intercept point       *)
(*-------------------------------------*)
FUNCTION pip(range:REAL):REAL;
VAR alpha,gamma2,t1,t2,bm:REAL;
BEGIN
alpha:=abs(abs(hook_az-hook_head)-180.0);
gamma2:=-1;
IF (range>sting_max) AND ( alpha<90) THEN
  BEGIN
  t1:=abs(sin(alpha*deg_rad));
  t2:=(range*t1)/sting_max;
  IF t2<=1.0 THEN
    gamma2:=180.0-(arcsin(t2)*rad_deg);
  IF gamma2 > 90.0 THEN
    bm:=(sting_max*abs(sin((180.0-alpha-gamma2)*deg_rad)))/t1;
  END;
IF (range<=sting_max) AND (range>=sting_min) THEN pip:=-1 ELSE
IF (gamma2<=90.0) OR (range<sting_min) THEN pip:=-2 ELSE
pip:=bm/hook_vel;
END;


(*-------------------------
(* clear hooked track
(*-------------------------*)
PROCEDURE erasemetricdata;
VAR x : INTEGER;
BEGIN
man_hook:=FALSE ;    (*rca*)
hook_a_track := FALSE ;
CONMOVECSR(44,1);
```

```
CONLINEOUT('          ',9);
FOR x := 2 to 10 DO
  BEGIN
  CONMOVECSR(44,x);
  CONLINEOUT('          ',8);
  END;
END ;

(*-------------------------
(* get the priority ranking
(*-------------------------*)
FUNCTION get_rank:INTEGER;
VAR count:INTEGER;
BEGIN
get_rank:=-1;
count:=1;
WHILE (hook_num<>rank_tbl[count].rank_num)AND(count <= stemp) DO
    count:=count+1;
IF count<=stemp THEN get_rank:=count;
END;

{--------------------------------------
{  Procedure hook_track_process
{   PURPOSE : locate hooked track in track table
{             compute x and y coordinates based on time lapse
{             compute azimuth, elevation, heading and range
{   INPUT   : hook_a_track ,hook_num
{             track_tbl
{   OUTPUT  : hook_rec
{--------------------------------------}
PROCEDURE hook_process ;
VAR twod_range         : REAL ;
    count              : INTEGER ;
    x_rel_fu,y_rel_fu  : REAL ;
    track_rec          : track_type ;
    time_laps          : BYTE ;
BEGIN
IF hook_a_track THEN
  BEGIN
  count := 1 ;
  WHILE (hook_num<>screen_tbl[count].screen_num)AND(count <= stemp) DO
    count:=count+1;
  IF (count <= stemp) THEN
    BEGIN
    WITH screen_tbl[count] DO
      BEGIN
      hook_weap   := screen_weap;
      hook_id     := screen_id;
      hook_vel    := screen_vel;
      hook_az     := screen_az;
      hook_range  := screen_3drange;
      hook_alt    := screen_alt;
      hook_head   := screen_head;
```

```
      ho k_el     := screen_el;
      hook_dhead  := screen_dhead;
      hook_ttc    := pip(hook_range);
      IF (hook_ttc = -1) AND (t_inrange.i = 0) THEN t_inrange.i:=
                    todclock.i;
      IF (pridisp<>0) THEN hook_rank:=get_rank;
      END ; { with screen rec }
    END
   ELSE
    erasemetricdata
  END ; { if .. then }
END ; { procedure hook_track_process }

(*-------------------------
(* quicker extrap for stinger
(*-------------------------*)
PROCEDURE sting_hook;
VAR twod_range        : REAL ;
    count             : INTEGER ;
    x_rel_fu,y_rel_fu : REAL ;
    track_rec         : track_type ;
    time_laps         : BYTE ;
BEGIN
IF hook_a_track THEN
  BEGIN
  count := 1 ;
  WHILE (hook_num<>track_tbl[count].track_num)AND(count <= max_track) DO
    count:=count+1;
  IF (count <= max_track) AND (track_tbl[count].active) THEN
    BEGIN
    track_rec := track_tbl[count] ;
    WITH track_rec DO
      BEGIN
      time_laps := (clock.b[1])-time_tag;
      x_rel_fu:=((pos_lsb*x_pos.i)+(x_vel*time_laps*time_vel_lsb)
                  -fu_coord[1]);
      y_rel_fu:=((pos_lsb*y_pos.i)+(y_vel*time_laps*time_vel_lsb)
                  -fu_coord[2]);
      hook_az:=pazimuth(y_rel_fu,x_rel_fu);
      twod_range := SQRT(x_rel_fu*x_rel_fu + y_rel_fu*y_rel_fu) ;
      hook_range:=SQRT(twod_range*twod_range+altitude.i*altitude.
                  i*pos_lsb_sq);
      IF (twod_range=0.0) THEN twod_range:=0.1;
      hook_el:=(ARCTAN((altitude.i*pos_lsb-fu_coord[3])/twod_range)
                  *rad_deg);
      END ; { with track_rec }
    END ;
  END;
END ; { procedure sting_hook }

{-------------- locate hooked track via hook ball ----------}
PROCEDURE ball_hook ;
var x,y,r,r1 : REAL ;
    count,thn : INTEGER ;
BEGIN
```

```
x := ball_x-screen_tbl[1].screen_x;
y := ball_y-screen_tbl[1].screen_y;
r := SQRT(x*x+y*y);
thn := screen_tbl[1].screen_num ;
FOR count := 2 to stemp DO
  BEGIN
  x := ball_x-screen_tbl[count].screen_x;
  y := ball_y-screen_tbl[count].screen_y;
  r1:= SQRT(x*x+y*y);
  IF r1 < r THEN
    BEGIN
    r := r1 ;
    thn := screen_tbl[count].screen_num ;
    END;
  END;
state := idle ;
hook_a_track := TRUE ;
man_hook := TRUE ;
hook_num := thn ;
t_hooknum:= thn;
t_inrange.i:=0;
t_lockon.i:=0;
t_hooktim.i:=todclock.i;
ball_x:=screen_fu_x;
ball_y:=screen_fu_y;
END;

(*-------------------------
(* sound alarm
(*-------------------------*)
PROCEDURE beepit;
BEGIN
OSSETPRIORITY(prog_pid,sound_pri,errorw);
beep;
OSSETPRIORITY(prog_pid,prog_pri,errorw);
END;

{---------------------------------------
{  Procedure table_manager
{   PURPOSE : locate active tracks in track table
{               call map_data
{               call memory_map
{             call hook_track_process
{             call display
{---------------------------------------}
PROCEDURE table_manager ;
VAR count : INTEGER ;
    time_laps : BYTE ;
BEGIN
count := 1 ;
stemp := 0 ;
tsound_alert := FALSE;
```

```
WHILE count <= max_track DO
  BEGIN
  IF track_tbl[count].active THEN
    BEGIN
    time_laps := (clock.b[1])-track_tbl[count].time_tag;
    IF time_laps > del_rate THEN
      BEGIN
      track_tbl[count].active := false;
      track_tbl[count].abell := 0;
      track_tbl[count].first_site := 0;
      END
     ELSE
      BEGIN
      track_rec := track_tbl[count] ;
      stemp:=stemp+1 ;
      map_data(stemp,count) ;
      END ;
    END ; { if .. then }
  count := count + 1 ;
  END ; { while }
IF (tsound_alert) AND (audible) THEN beepit;
IF state = ball_active THEN ball_hook;
IF (pridisp<>0) THEN
  BEGIN
  prioritize;
  rank_priority;
  END;
hook_process ;
display;
END ; { procedure hook_track_process }

(*-------------------------
(* unsound alarm
(*-------------------------*)
PROCEDURE alert_off;
VAR x : INTEGER;
BEGIN
FOR x := 1 to max_track DO
  IF ((track_tbl[x].active) and (track_tbl[x].abell = 1)) THEN
    track_tbl[x].abell := 2;
tsound_alert := FALSE ;
END;

(*-------------------------
(* ranking number for prio
(*-------------------------*)
FUNCTION get_rank_num(x:INTEGER):INTEGER;
VAR count:INTEGER;
BEGIN
get_rank_num:=-1;
IF x<=stemp THEN get_rank_num:=rank_tbl[x].rank_num;
END;
```

```
(*-------------------------
(* data capture
(*-------------------------*)
PROCEDURE writeit;
VAR x : INTEGER;
BEGIN
FOR x := 1 TO stemp DO captureit(screen_tbl[x]);
END;


{--------------------------------------
{  Procedure fkey_manager
{   PURPOSE : handle interrupts from keyboard
{             perform necessary keyboard functions
{   INPUT   : char_in
{             in_process
{   OUTPUT  : hook_a_track
{             hook_number
{             cage_bit
{             led_bit
{--------------------------------------}
PROCEDURE fkey_manager ;
VAR char_in  :  char_byte ;
    count    :  INTEGER ;
    test     :  BOOLEAN ;
BEGIN
char_in.c := CONCHARIN ;
CASE state OF
   num_pend :BEGIN
             IF (char_in.b in [30H .. 39H ]) AND (ccount <= 2) THEN
              BEGIN
              CONMOVECSR(45+ccount,13);
              CONCHAROUT(char_in.c);
              ccount := ccount + 1 ;
              kbuff[ccount] := char_in.b-30H ;
              END
             ELSE
              IF char_in.b = cr THEN
                BEGIN
                FOR count := 1 to ccount DO
                  t_hook_num := t_hook_num*10 + kbuff[count];
                hook_a_track := ccount <> 0 ;
                man_hook:=hook_a_track;
                IF NOT rank THEN hook_num := t_hook_num ELSE
                hook_num:=get_rank_num(t_hook_num);
                t_hooknum:= hook_num ;
                t_inrange.i:=0;
                t_lockon.i:=0;
                t_hooktim.i:=todclock.i;
                state := idle ;
                CONMOVECSR(30,13);CONLINEOUT('                    ',19);
                IF NOT hook_a_track THEN erasemetricdata;
                END
```

```
            END;
      idle,
ball_pend:CASE char_in.b OF
              196 : ball_y := ball_y + 5;
              198 : ball_x := ball_x - 5;
              199 : ball_x := ball_x + 5;
              197 : ball_y := ball_y - 5;
              206 : ball_y := ball_y + 40;
              208 : ball_x := ball_x - 40;
              209 : ball_x := ball_x + 40;
              207 : ball_y := ball_y - 40;
          68h,48h : state:=ball_active;
          69h,49h : IF meters_pixel > 40 THEN meters_pixel:=
                       meters_pixel DIV 2 ;
          6fh,4fh : IF meters_pixel < 640 THEN meters_pixel:=
                       meters_pixel * 2 ;
          75h,55h : screen_fu_y := screen_fu_y-10 ;
          64h,44h : screen_fu_y := screen_fu_y+10 ;
          6ch,4ch : screen_fu_x := screen_fu_x-10 ;
          72h,52h : screen_fu_x := screen_fu_x+10 ;
          7ah,5ah : led_bit := (led_bit + 1) MOD 8;
             21h : stop:=TRUE;
          70h,50h : BEGIN
                    update_parm(TRUE);
                    IF demo_mode THEN
                      iowrapup
                     ELSE
                      BEGIN
                      deactivateradar;
                      fake_radar:=FALSE;
                      IF NOT iaok THEN comm_init(FALSE);
                      END;
                    END;
          73h,53h : BEGIN
                    hook_a_track := stemp <> 0 ;
                    man_hook:=hook_a_track;  (*rca*)
                    IF hook_a_track THEN
                     REPEAT
                        hook_idx := (hook_idx MOD max_track) + 1 ;
                     UNTIL track_tbl[hook_idx].active ;
                    hook_num := track_tbl[hook_idx].track_num ;
                    t_hooknum:= hook_num ;
                    t_inrange.i:=0;
                    t_lockon.i:=0;
                    t_hooktim.i:=todclock.i
                    END ;
          6eh,4eh : BEGIN
                    ccount := 0 ;
                    t_hook_num := 0 ;
                    state := num_pend ;
                    rank:=FALSE;
                    CONMOVECSR(30,13);CONLINEOUT('HOOK NUMBER ?: ',15);
                    END ;
          30h,31h,
          32h,33h,
```

```
34h,35h,
36h,37h,
38h,39h : IF (pridisp<>0) THEN
            BEGIN
            ccount := 1 ;
            kbuff[ccount]:=char_in.b-30h;
            t_hook_num := 0 ;
            state := num_pend ;
            rank:=TRUE;
            CONMOVECSR(30,13);
            CONLINEOUT('PRIORITY RANK: ',15);
            CONCHAROUT(char_in.c);
            END ;
63h,43h : BEGIN
          screen_fu_x := win_dim DIV 2 ;
          screen_fu_y := win_dim DIV 2 ;
          END;
     20h: alert_off;
74h,54h : BEGIN
          CONMOVECSR(30,0);
          IF NOT capture THEN
            BEGIN
            testfn:=capname;
            testfn[1]:=CHR(17);
            error1 := OSATTACH(testfn,3,fonres,3,error);
            OSDETACH(error1,error2);
            IF error <> 0 THEN
              conlineout('DSKERR',6)
             ELSE
              BEGIN
              REWRITE(capfil,''w'dat'data-text~');
              conlineout('CAPTURE',7);
              capful:=0;
              mempos:=0; (*28apr86*)
              capture := not capture;
              END
            END
           ELSE
            BEGIN
            error2:=OSWAIT(csid,1000,error1);
            capture := not capture;
            PQCLOSE(capfil);
            conlineout('       ',7);
            END;
          END;
6dh,4dh : BEGIN
          mils:=(mils+1) MOD 2;
          CONMOVECSR(52,4);
          CASE mils OF
            0:CONLINEOUT('deg',3);
            1:CONLINEOUT('mil',3);
            END; (* case *)
          CONMOVECSR(52,5);
```

```
                    CASE mils OF
                      0:CONLINEOUT('deg',3);
                      1:CONLINEOUT('mil',3);
                      END; (* case *)
                    END;
            76h,56h : BEGIN
                    mph:=NOT mph;
                    CONMOVECSR(52,7);
                    IF mph THEN
                      CONLINEOUT('kts',3)
                     ELSE
                      CONLINEOUT('m/s',3);
                    END;
             66h,46h: BEGIN
                    IF demo_mode THEN
                      BEGIN
                      fake_radar:=NOT fake_radar;
                      IF fake_radar THEN
                        activateradar
                       ELSE
                        deactivateradar;
                      END;
                    END;
             65h,45h: evaluate;
                 03h: BEGIN
                    deactivateradar;
                    OSDELETEPROCESS(time_pid,errorw);
                    OSDELETEPROCESS(sp,errorw);
                    END;
          END ; { case }
  END; { case }
END ; { procedure fkey_manager }


{----------- main program -----------}
BEGIN
initialize ;
REPEAT
  WHILE conkeypressed DO fkey_manager ;
  table_manager ;
  IF capture THEN writeit;
UNTIL stop ;
bfont := WINSETFONT(lfont,fonrec,0);
CONRESETDISPLAY;
OSEXIT(0);
END
```

```
$NOLIST

$COMPACT(-const in code-)

MODULE iointerface ;

$INCLUDE ('w'incs'common.inc-text-)
$INCLUDE ('w'incs'conpas.inc-text-)
$INCLUDE ('w'incs'ospasprocs.inc-text-)
$INCLUDE ('w'incs'ospastypes.inc-text-)
$INCLUDE ('w'incs'windowprocs.inc-text-)
$INCLUDE ('w'incs'windowtypes.inc-text-)

(*-------------------------------------------------*)
(*     INTERFACE SECTION                           *)
(*                                                 *)
(*-------------------------------------------------*)

PUBLIC tablemgr ;
CONST       syn1       = 0AAH;
            syn2       = 55H;
            io_buff_lng = 11;

VAR         io_buffer  : ARRAY [1..io_buff_lng] of BYTE ;
            io_pid     : WORD ;
            fonrec     : FONTINFORECORD ;
            lfont,bfont: fontpointer ;
            dumptr     : windowregionptr ;
            pe;me      : WORD;

PUBLIC tablemgr ;
PROCEDURE msg_handler ;

PUBLIC iointerface ;
PROCEDURE comm_init(man_dial:BOOLEAN) ;
PROCEDURE handle_it ;
PROCEDURE iowrapup;

PUBLIC iointerface ;
VAR   iaok : BOOLEAN ;


PRIVATE iointerface  ;
CONST
      devname    = ' 'modem' ;
      io_length = 11;
      fifolength= 250 ;
      io_delay  = 300 ;
TYPE
      stattype  = ARRAY [1..15] of byte ;
      fifoptr   = ^fifobuffer ;
      fifobuffer= ARRAY [1..fifolength] of BYTE ;
      fifotype  = RECORD
                    mode : CHAR ;
```

```
                    ptr   : fifoptr ;
                    length  : WORD ;
                  END;
      totype    = RECORD
                    mode   : CHAR ;
                    val1   : WORD;
                    val2   : WORD;
                  END ;


VAR
      fifo        : fifotype ;
      in_buff     : fifobuffer ;
      modemid     : WORD ;
      ch,kb       : CHAR ;
      parblock    : PACKED ARRAY [1..9] of CHAR ;
      pathname    : PACKED ARRAY [1..7] of CHAR ;
      reserved    : BYTE ;
      error       : WORD ;
      nlength     : INTEGER ;
      actual      : INTEGER ;
      io_pos      : INTEGER ;
      statrec     : stattype ;
      io_state    : INTEGER ;
      csum        : BYTE ;
      timout      : totype ;
      ip          : WORD ;
      dptr        : windowregionptr;
      thur        : WORD;
(*-------------------------------------*)
(*   IOWRAPUP shuts down modem port    *)
(*-------------------------------------*)
PROCEDURE iowrapup;
BEGIN
iaok := FALSE;
OSDELETEPROCESS(ip,error);
parblock[1] := CHR(5);
OSSETSTATUS(modemid,parblock,1,error);
parblock[1] := CHR(42);
OSSETSTATUS(modemid,parblock,1,error);
OSCLOSE(modemid,error);
OSDETACH(modemid,error);
END;

(*----------------------
(* comm status
(*----------------------*)
PROCEDURE csys ;
BEGIN
WINSETALTERNATEWINDOW(NIL);
CONMOVECSR(45,24);
CONLINEOUT('COMM STAT: ',11);
```

```
END;
(*-----------------------
(* comm err r
(*-----------------------*)
PROCEDURE commerr(error:WORD);
VAR dfont:fontpointer;
BEGIN
IF (error = 403) THEN
 pe := pe + 1
ELSE
 BEGIN
 dfont := WINSETFONT(lfont,fonrec,0);
 csys ;
 CASE error of
  231:CONLINEOUT    ('NOT ACTIVE      ',16);
  998:CONLINEOUT    ('ACTIVATING      ',16);
  999:CONLINEOUT    ('ON LINE         ',16);
  221:CONLINEOUT    ('BAD CONNECT     ',16);
  400:CONLINEOUT    ('NO ANSWER       ',16);
  401:CONLINEOUT    ('TIME-OUT        ',16);
  402:CONLINEOUT    ('LOST CARRIER    ',16);
  406:CONLINEOUT    ('INVALID NUMBER  ',16);
  OTHERWISE CONHEXOUT(error);
  END; (*case*)
 WINSETALTERNATEWINDOW(dumptr);
 dfont := WINSETFONT(bfont,fonrec,0);
 IF not ( error in [0ffffh,997,998,999,403] ) THEN iaok := false;
 END;
END;

(*-----------------------------------------------*)
(*  IOHANDLER builds IO_BUFFER                   *)
(*-----------------------------------------------*)

PROCEDURE io_handler ;
VAR  x : INTEGER ;
     by: BYTE ;
     lng: WORD;
BEGIN
lng:=statrec[8];
actual := OSREAD(modemid,in_buff,lng,error);
IF error <> 0 THEN commerr(error);
FOR x := 1 to actual DO
  BEGIN
  by := in_buff[x];
  CASE io_state OF
    0:IF by = syn1 THEN io_state := 1 ;
    1:IF by = syn2 THEN
        io_state := 2
       ELSE
        BEGIN
        csum := 0 ;
```

```
        io_state := 0 ;
        END;
    2:BEGIN
      IF io_pos = io_length THEN
        BEGIN
        io_state := 0 ;
        io_pos := 0 ;
        IF csum = by THEN
          msg_handler
         ELSE
          me := me+1;
        csum := 0;
        END
       ELSE
        BEGIN
        csum := csum + by ;
        io_pos := io_pos + 1 ;
        io_buffer[io_pos] := by ;
        END;
      END;
    END; (* case  *)
  END; (* for..next *)
END; (* proc *)


(*-------------------------------------*)
(* CHAR_DET sees if anything in FIFO buffer?*)
(*          also checks for comm errors     *)
(*-------------------------------------*)

FUNCTION char_det : BOOLEAN ;
VAR x : INTEGER ;
BEGIN
OSGETSTATUS(modemid,statrec,15,error);
IF statrec[11] <> 2 THEN error := 402 ;
char_det :=  (statrec[8] <> 0) AND (error = 0) ;
if statrec[8] > pe then pe:=statrec[8];
IF error <> 0 THEN commerr(error);
END;



(*----------------------
(* initialize commo
(*----------------------*)

PROCEDURE comm_init (man_dial:BOOLEAN);
VAR dfont:fontpointer;
BEGIN
thur:=0;
io_state := 0;
```

```
io_pos := 0;
csum := 0;
iaok:=FALSE;
pe := 0;
me := 0;
commerr(998);
OSDELETEPROCESS(ip,error);
iowrapup;
(*---------------- attach the modem ----------------*)
pathname := devname ;
pathname[1] := CHR(6) ;
reserved := 0 ;
modemid := OSATTACH(pathname,oldfilemode,reserved,1,error);
IF error <> 0 THEN commerr(error);
(*----------------- open modem -------------*)
OSOPEN(modemid,1,error);
IF error <> 0 THEN commerr(error);
(* ------------- init modem ------------- *)
parblock[1] := CHR(1) ;  (* mode *)
parblock[2] := CHR(1) ;  (* async *)
parblock[3] := CHR(8) ;  (* 8 dtat bits *)
parblock[4] := CHR(1) ;  (* 1 stop bits *)
parblock[5] := CHR(1) ;  (* even parity *)
OSSETSTATUS(modemid,parblock,5,error);
IF error <> 0 THEN commerr(error);
(*-------  new fifo --------- *)
fifo.mode := CHR(4);
NEW(fifo.ptr);
fifo.length := fifolength ;
OSSETSTATUS(modemid,fifo,5,error);
IF error <> 0 THEN commerr(error);
(*--------------- baud rate -----------------*)
parblock[1] := CHR(7) ;  (* mode *)
parblock[2] := CHR(7) ;  (* 1200 *)
OSSETSTATUS(modemid,parblock,2,error);
ip := OSFORKPROCESS(handle_it,40,TRUE,2000,error);   (* 20 - old serial *)
END;


(*-----------------------------------------------*)
(* HANDLE_IT  task used to retreive radar data  *)
(*            or establish comm link            *)
(*-----------------------------------------------*)
PROCEDURE handle_it ;
BEGIN
REPEAT
 WHILE iaok DO
   BEGIN
   OSDELAY(io_delay);
   WHILE char_det DO io_handler
   END;
 WHILE NOT iaok DO
   BEGIN
```

```
    OSDELAY(io_delay);
    parblock[1] := CHR(5);
    OSSETSTATUS(modemid,parblock,1,error);
    parblock[1] := CHR(42);
    OSSETSTATUS(modemid,parblock,1,error);
    (*-------------- answer mode -------------*)
    parblock[1] := CHR(43) ;  (* mode *)
    parblock[2] := CHR(2) ;  (* 9600 *)
    OSSETSTATUS(modemid,parblock,2,error);
    (*-------------- off hook ----------------*)
    parblock[1] := CHR(41) ;  (* mode *)
    OSSETSTATUS(modemid,parblock,1,error);
    (*-------------- set time out limits ------*)
    timout.mode := CHR(2);
    timout.val1 := 3000;
    timout.val2 := 10000;
    OSSETSTATUS(modemid,timout,5,error);
    (*-------------- data mode ---------------*)
    parblock[1] := CHR(6) ;
    OSSETSTATUS(modemid,parblock,1,error);
    IF (error = 0) THEN
      BEGIN
      commerr(999);
      iaok := true;
      END;
    END;
UNTIL FALSE ;
END;
```

```
$NOLIST
$COMPACT(-const in code-)

MODULE parminterface ;

$INCLUDE ('w'incs'common.inc-text-)
$INCLUDE ('w'incs'conpas.inc-text-)
$INCLUDE ('w'incs'ospasprocs.inc-text-)
$INCLUDE ('w'incs'ospastypes.inc-text-)
$INCLUDE ('w'incs'windowprocs.inc-text-)
$INCLUDE ('w'incs'windowtypes.inc-text-)

(*-----------------------------------*)
(*  INTERFACE SECTION                *)
(*-----------------------------------*)

PUBLIC CommonTypes;
TYPE
  StringDescriptor = RECORD
                       len  : Word;
                       max  : Word;
                       dummy: Byte;
                       chars: ARRAY [1..65535] OF Char;
                     END;
  StringPtr = ^StringDescriptor;


PUBLIC StringProcs;
FUNCTION  NewString(maxLength: Word): StringPtr;
PROCEDURE AppendChar(dest: StringPtr; ch: Char);
PROCEDURE FreeString(VAR str: StringPtr);
FUNCTION  StringToReal(str: StringPtr; VAR converted: Boolean): LongReal;

PUBLIC tablemgr ;
CONST
     maxentry = 16 ;
TYPE
  string8       =  packed array [1..8] of char;
  erectype      =  RECORD
                     xpos: INTEGER ;
                     ypos: INTEGER ;
                     mlng: BYTE ;
                     val : string8;
                     editt: byte;
                     END;
  coord_type    =  ARRAY [1..3] of LONGINT ;     { LSB of 2.34375  }

VAR
  erec          :  ARRAY [0..maxentry] of erectype ;
  grid_mag_north: REAL ;
  dlrp_coord    :  coord_type ;        { DLRP x, y, and z coordinates       }
  fu_coord      :  coord_type ;        { Fire Unit x, y, and z coordinates  }
```

```
  alert_range  :  INTEGER ;
  optr         :  windowr gionptr ;
  opoint       :  point ;
  orect        :  rectangle ;
  fonrec       :  FONTINFORECORD ;
  lfont,bfont  :  fontpointer ;
  audible      :  BOOLEAN ;
  auto_hook    :  BOOLEAN ;  (*rca*)
  demo_mode    :  BOOLEAN ;
  pridisp      :  BYTE ;
  startps      :  REAL ;
  endps        :  REAL ;
  pvel         :  REAL ;
  palt         :  REAL ;

PUBLIC parminterface ;
PROCEDURE update_parm(tf:BOOLEAN) ;

PUBLIC tablemgr;
PROCEDURE beepit;


PRIVATE parminterface ;
CONST
      ledit    = 0;
      medit    = 1;
      pedit    = 2;
      redit    = 3;
      bedit    = 4;
      iedit    = 5;
      pmedit   = 6;

TYPE
      estringtype = RECORD
                      chars : string8 ;
                      len : integer;
                      END;


VAR  estring : estringtype ;
     quit    : BOOLEAN ;
     entry   : INTEGER ;
     neg     : BOOLEAN ;
     dfont   : fontpointer;

(*-------------------------------------------*)
(*   start of regular parameter menu         *)
(*-------------------------------------------*)


(*---------------------
(* position the cursor
(*--------------------*)
```

```
PROCEDURE poscursor ;
BEGIN
CONMOVECSR(erec[entry].xpos, rec[entry].ypos);
END;


(*---------------------
(* keyboard input of string
(*--------------------*)
PROCEDURE getstring ;
VAR by : BYTE ;
    x:integer;
BEGIN
CONDEFCSR(TRUE);
estring.chars:='          ';
estring.len:=0;
repeat
  by := ORD(CONCHARIN) ;
  IF (NOT (by in [13,27])) AND (estring.len < erec[entry].mlng) THEN
    BEGIN
    WRITE(CHR(by));
    estring.len:=estring.len+1;
    estring.chars[estring.len]:=chr(by);
    END;
UNTIL (by in [13,27]);
quit := (by = 27);
CONDEFCSR(FALSE);
IF estring.len <> 0 THEN
  for x:= estring.len+1 to erec[entry].mlng do concharout(' ');
END;

(*--------------------
(* real number validate
(*--------------------*)
FUNCTION arrtoreal(s:string8;var b:boolean):real;
var t:stringptr;x:integer;
BEGIN
t:=newstring(8);
x:=1;
while s[x] <> ' ' do
  begin
  appendchar(t,s[x]);
  x:=x+1;
  end;
arrtoreal:=stringtoreal(t,b);
freestring(t);
END;

(*---------------------
(* mills  validate
(*--------------------*)
FUNCTION mlcheck(s:string8):boolean;
var r:real;b:boolean;
```

```
BEGIN
r:=arrtoreal(s,b);
if b then b:=(r<=6400)and(r>=0);
mlcheck:=b;
END;

(*---------------------
(* priority mills  validate
(*---------------------*)
FUNCTION pmlcheck(s:string8):boolean;
var r:real;b:boolean;
BEGIN
r:=arrtoreal(s,b);
if b then b:=(r<=6400)and(r>=-6400);
pmlcheck:=b;
END;



(*-------------------------*)
(* AOK validates entry     *)
(*-------------------------*)

FUNCTION aok : BOOLEAN ;
VAR taok : BOOLEAN ;
    tlint:longint;treal:real;
    tstring:string8;
BEGIN
taok := TRUE ;
if estring.len <> 0 then
case erec[entry].editt of
     iedit:treal:=arrtoreal(estring.chars,taok);
     ledit:treal:=arrtoreal(estring.chars,taok);
     bedit:taok:=(estring.chars[1] in ['0','1']);
     pedit:taok:=(estring.chars[1] in ['0'..'2']);
     medit:taok:=mlcheck(estring.chars);
     pmedit:taok:=pmlcheck(estring.chars);
     redit:treal:=arrtoreal(estring.chars,taok);
     end;
aok:=taok;
END ;

(*---------------------
(* save entry
(*---------------------*)
PROCEDURE saveit ;
var x : integer;
BEGIN
if estring.len<>0 then
  begin
  erec[entry].val:='         ';
  for x:=1 to estring.len do erec[entry].val[x]:= estring.chars[x];
  end;
```

```
END ;

(*---------------------
(* adjust cursor
(*--------------------*)
PROCEDURE adjpos ;
BEGIN
entry := (entry + 1) MOD maxentry ;
END ;


(*---------------------
(* init screen and table
(*--------------------*)
PROCEDURE initialize ;
VAR x : INTEGER ;

BEGIN
quit := FALSE ;
WINCOPYREMOTERECTANGLE(NIL,optr,orect,opoint,0);
dfont := WINSETFONT(lfont,fonrec,0);
CONRESETDISPLAY;
CONDEFCSR(FALSE);
CONMOVECSR(0,1);
CONLINEOUT('          *** GRID DISPLAY UNIT PARAMETER MENU ***',50);
CONMOVECSR(0,3);
CONLINEOUT(' DATA LINK REFERENCE POINT ( meters )..E:        N:
                    Alt:',63);
CONMOVECSR(0,5);
CONLINEOUT(' FIRE UNIT LOCATION  ( meters )........E:        N:
                    Alt:',63);
CONMOVECSR(0,7); CONLINEOUT(' MAGNETIC NORTH DECLINATION TO GRID NORTH
(Mils)....E:',54); CONMOVECSR(0,9); CONLINEOUT(' ALERT RANGE ( meters
)....... :',32); CONMOVECSR(0,11); CONLINEOUT(' AUDIBLE ALERT
(0-off/1-on)... :',32); CONMOVECSR(0,13); CONLINEOUT(' AUTOMATIC HOOK
(0-off/1-on).. :',32); CONMOVECSR(0,15); CONLINEOUT(' DEMO MODE
(0-off/1-on)..... :',32); CONMOVECSR(0,17); CONLINEOUT(' PRIORITY
(0-off/1-on).................:',40); CONMOVECSR(0,19); CONLINEOUT(' START
OF PRIORITY SECTOR (mils).....:',40); CONMOVECSR(0,20);
CONLINEOUT(' END OF PRIORITY SECTOR (mils).....:',40);
CONMOVECSR(0,21);
CONLINEOUT(' VELOCITY CUTOFF.....................:',40);
CONMOVECSR(0,22);
CONLINEOUT(' ALTITUDE CUTOFF.....................:',40);
CONMOVECSR(0,23);
CONLINEOUT(' ** <esc> to return to Monitor **',33);
entry := 0 ;
erec[0].xpos := 42 ;
```

```
erec[0].ypos := 3 ;
erec[0].mlng := 7 ;
erec[0].editt:= ledit;
erec[1].xpos := 52 ;
erec[1].ypos := 3 ;
erec[1].mlng := 7 ;
erec[1].editt:= ledit;
erec[2].xpos := 64 ;
erec[2].ypos := 3 ;
erec[2].mlng := 7 ;
erec[2].editt:= ledit;
erec[3].xpos := 42 ;
erec[3].ypos := 5 ;
erec[3].mlng := 7 ;
erec[3].editt:= ledit;
erec[4].xpos := 52 ;
erec[4].ypos := 5 ;
erec[4].mlng := 7 ;
erec[4].editt:= ledit;
erec[5].xpos := 64 ;
erec[5].ypos := 5 ;
erec[5].mlng := 7 ;
erec[5].editt:= ledit;
erec[6].xpos := 55 ;
erec[6].ypos := 7 ;
erec[6].mlng := 7 ;
erec[6].editt:= pmedit;
erec[7].xpos := 33 ;
erec[7].ypos := 9 ;
erec[7].mlng := 5 ;
erec[7].editt:= iedit;
erec[8].xpos := 33 ;
erec[8].ypos := 11 ;
erec[8].mlng := 1 ;
erec[8].editt:= bedit;
erec[9].xpos := 33 ;
erec[9].ypos := 13 ;
erec[9].mlng := 1 ;
erec[9].editt:= bedit;
erec[10].xpos := 33 ;
erec[10].ypos := 15 ;
erec[10].mlng := 1 ;
erec[10].editt:= bedit;
erec[11].xpos := 41 ;
erec[11].ypos := 17 ;
erec[11].mlng := 1 ;
erec[11].editt:= bedit;
erec[12].xpos := 41 ;
erec[12].ypos := 19 ;
erec[12].mlng := 7 ;
erec[12].editt:= medit;
erec[13].xpos := 41 ;
erec[13].ypos := 20 ;
```

```
erec[13].mlng := 7 ;
erec[13].editt:= medit;
erec[14].xpos := 41 ;
erec[14].ypos := 21 ;
erec[14].mlng := 7 ;
erec[14].editt:= redit;
erec[15].xpos := 41 ;
erec[15].ypos := 22 ;
erec[15].mlng := 7 ;
erec[15].editt:= redit;
FOR x := 0 to maxentry-1 DO
  BEGIN
  CONMOVECSR(erec[x].xpos,erec[x].ypos);
  conlineout(erec[x].val,erec[x].mlng);
  END;
END;

(*---------------------
(* save all parameters
(*--------------------*)
PROCEDURE saveparms ;
VAR x : INTEGER ;b:boolean;
BEGIN
dlrp_coord[2] := ltrunc(arrtoreal(erec[1].val,b)) ;
fu_coord[2] := ltrunc(arrtoreal(erec[4].val,b))-dlrp_coord[2];
dlrp_coord[1] := ltrunc(arrtoreal(erec[0].val,b));
fu_coord[1] := ltrunc(arrtoreal(erec[3].val,b))-dlrp_coord[1];
dlrp_coord[3] := ltrunc(arrtoreal(erec[2].val,b));
fu_coord[3] := ltrunc(arrtoreal(erec[5].val,b))-dlrp_coord[3];
alert_range:=trunc(arrtoreal(erec[7].val,b));
audible := (erec[8].val[1] = '1');
auto_hook := (erec[9].val[1] = '1');
demo_mode := (erec[10].val[1] = '1');
pridisp  :=  (ord(erec[11].val[1])-30h);
startps := arrtoreal(erec[12].val,b)/17.77777;
endps := arrtoreal(erec[13].val,b)/17.77777;
pvel := arrtoreal(erec[14].val,b);
palt := arrtoreal(erec[15].val,b);
grid_mag_north := arrtoreal(erec[6].val,b)/17.77777;
IF demo_mode THEN
  BEGIN
  WINSETALTERNATEWINDOW(optr);
  CONMOVECSR(56,24);
  CONLINEOUT('DEMO MODE       ',16);
  WINSETALTERNATEWINDOW(NIL);
  END;
dfont := WINSETFONT(bfont,fonrec,0);
WINCOPYREMOTERECTANGLE(optr,NIL,orect,opoint,0);
CONDEFCSR(FALSE);
END;
```

```
(*-----------------------
(* alarm and display old val
(*----------------------*)
PROCEDURE writeerror ;
BEGIN
beepit;
CONMOVECSR(erec[entry].xpos,erec[entry].ypos);
conlineout(erec[entry].val,erec[entry].mlng);
END;


(*-----------------  main proc ------------*)
PROCEDURE syspar;
BEGIN
initialize ;
WHILE NOT quit DO
  BEGIN
  poscursor ;
  getstring ;
  IF aok THEN
    BEGIN
    saveit ;
    adjpos ;
    END
   ELSE
    writeerror ;
  END ;
saveparms ;
END ;


PROCEDURE update_parm(tf:boolean) ;
BEGIN
IF tf THEN syspar;
END;
```

```
$NOLIST
$COMPACT(-const in code-)

MODULE stingerhandler ;

$INCLUDE ('w'incs'common.inc-text-)
$INCLUDE ('w'incs'conpas.inc-text-)
$INCLUDE ('w'incs'ospasprocs.inc-text-)
$INCLUDE ('w'incs'ospastypes.inc-text-)

(*-------------------------------------------------------------*)
(*      INTERFACE SECTION                                      *)
(*-------------------------------------------------------------*)

PUBLIC tablemgr ;
CONST  syn1         = 0AAH ;
       syn2         = 55H ;
TYPE
       byte_int     = RECORD
                        CASE BOOLEAN OF
                          TRUE :(b:ARRAY [1..2] of BYTE) ;
                          FALSE:(i:INTEGER);
                      END ;
       cal_type     = ARRAY[0..4,0..4] of REAL ;
       clock_type   =  RECORD
                          CASE BOOLEAN OF
                            TRUE : (b:ARRAY [1..4] of BYTE) ;
                            FALSE: (i:LONGINT);
                       END ; ( record )

VAR
       todclock     : clock_type ;
       capture      : BOOLEAN;
       hook_x       : INTEGER;
       hook_y       : INTEGER;
       hook_z       : INTEGER;
       in_range_bit: INTEGER ;
       status_bit   : INTEGER ;
       led_bit      : INTEGER ;
       hook_a_track: BOOLEAN ;
       hook_az      : REAL;
       hook_el      : REAL;
       hook_range   : REAL    ;
       sting_az     : byte_int ;
       sting_el     : byte_int ;
       sting_on     : BOOLEAN ;
       cal_tbl      : cal_type;
       el_off       : REAL;
       lockon       : BOOLEAN;
       grid_mag_north: REAL;


PUBLIC tablemgr ;
```

```
PROCEDURE hook_process ;
PROCEDURE sting_hook ;

PUBLIC stingerhandler ;
PROCEDURE sting_init ;
PROCEDURE sting_it ;

PUBLIC stingerhandler;
VAR sp : WORD ;


PRIVATE stingerhandler ;
CONST devname      = ' 'serial' ;
      sting_length = 7;
      fifolength   = 250 ;
      sting_delay  = 30 ;
      az_lsb       = 0.125;
      el_lsb       = 0.3515625;
      sting_max_rng= 5000.0;
      sting_min_rng= 1000.0;
      poll_max     = 40;    (*28jan86*)
      deg_rad      = 0.01745329252;

TYPE
      stattype     = ARRAY [1..15] of byte ;
      fifoptr      = ^fifobuffer ;
      fifobuffer   = ARRAY [1..fifolength] of BYTE ;
      fifotype     = RECORD
                       mode : CHAR ;
                       ptr  : fifoptr ;
                       length  : WORD ;
                     END;
VAR
      az_error     : byte_int ;
      el_error     : byte_int ;
      fifo         : fifotype ;
      stingfifo    : fifobuffer ;
      serialid     : WORD ;
      ch,kb        : CHAR ;
      parblock     : PACKED ARRAY [1..9] of CHAR ;
      pathname     : PACKED ARRAY [1..8] of CHAR ;
      reserved     : BYTE ;
      error        : WORD ;
      actual       : INTEGER ;
      sting_start  : INTEGER ;
      row,col      : INTEGER ;
      sting_buffer : ARRAY [1..sting_length] of BYTE ;
      statrec      : stattype ;
      stop         : BOOLEAN ;
      sting_state  : INTEGER ;
      csum         : BYTE ;
      sting_io_out : ARRAY [1..7] of BYTE ;
```

```
      sting_io_in   : ARRAY [1..sting_length] of BYTE ;
      sting_stat    : INTEGER ;
      el_off_i      : INTEGER;
      poll_cnt      : BYTE;     (*28jan86*)
      sendmsg       : BOOLEAN;


(*------------------------------
(*  calibrate stinger az/el
(*-----------------------------*)
PROCEDURE calibrate(VAR az,el:INTEGER);
VAR idx : INTEGER;
    ch  : REAL ;

FUNCTION el_idx(ce:REAL):INTEGER;
BEGIN
IF ce<2.5 THEN el_idx:=0 ELSE
IF ce<7.5 THEN el_idx:=1 ELSE
IF ce<12.5 THEN el_idx:=2 ELSE
IF ce<17.5 THEN el_idx:=3 ELSE
el_idx:=4;
END;

BEGIN
el:=el+el_off_i;
idx:=el_idx(el*az_lsb);
ch:=az*az_lsb*deg_rad;
az:=az + TRUNC((cal_tbl[0,idx]           +   (* deviation a          *)
                cal_tbl[1,idx]*sin(ch)   +   (* deviation b          *)
                cal_tbl[2,idx]*cos(ch)   +   (* deviation c          *)
                cal_tbl[3,idx]*sin(2*ch) +   (* deviation d          *)
                cal_tbl[4,idx]*cos(2*ch) +   (* deviation e          *)
                grid_mag_north)/az_lsb);     (* mag north deviation *)
IF az >= 2880 THEN az := az-2880 ELSE
IF az < 0 THEN az := az+2880;
END;


(*------------------------------
(*  serial i/o error
(*-----------------------------*)
PROCEDURE commerr(error:WORD);
BEGIN
IF NOT ( error in [997,998,999,403] ) THEN wrapup;
END;



(*------------------------------
(*  serial fifo check
(*-----------------------------*)
FUNCTION char_det : BOOLEAN ;
VAR x : INTEGER ;
```

```
BEGIN
OSGETSTATUS(serialid,statrec,15,error);
IF statrec[11] <> 2 THEN error := err r + statrec[11] + 1 ;
IF error <> 0 THEN commerr(error);
char_det :=  (statrec[8] <> 0) AND (error = 0) ;
END;

(*------------------------------
(*  serial shutdown
(*------------------------------*)
PROCEDURE wrapup;
BEGIN
sting_on := FALSE ;
poll_cnt:=poll_max;
parblock[1] := CHR(5);
OSSETSTATUS(serialid,parblock,1,error);
OSCLOSE(serialid,error);
OSDETACH(serialid,error);
END;

(*------------------------------
(*  comput LED blinking rate
(*------------------------------*)
FUNCTION az_rate(error:integer):integer;
VAR trate:integer;
BEGIN
IF ABS(error) <= 16 THEN trate:=0 ELSE
trate:=ROUND(60+(185-(ABS(error DIV 16) * 2.05)));
if error < 0 THEN
 trate := trate * 256;
az_rate:=trate;
END;

(*------------------------------
(*  comput LED blinking rate
(*------------------------------*)
FUNCTION el_rate(error:integer):integer;
VAR trate:integer;
BEGIN
IF ABS(error) <= 16 THEN trate:=0 ELSE
trate:=ROUND(60+(185-(ABS(error DIV 16) * 4.1)));
if error > 0 THEN
  trate := trate * 256;
el_rate:=trate;
END;


{-----------------------------------
{  Procedure send_stinger
{   PURPOSE : builds stinger message in byte format
{   INPUT   : az_error
{             el_error
{             in_range_bit
```

```
{              led_bit
{              cage_bit
{   OUTPUT   : sting_sting_out
{-------------------------------------}
PROCEDURE send_stinger ;
VAR tempa,tempe:byte_int;
BEGIN
tempa.i:=az_rate(az_error.i);
tempe.i:=el_rate(el_error.i);
sting_io_out[3]:=tempa.b[2];
sting_io_out[4]:=tempa.b[1];
sting_io_out[5]:=tempe.b[2];
sting_io_out[6]:=tempe.b[1];
sting_io_out[7] := in_range_bit + led_bit ;
OSWRITE(serialid,sting_io_out,7,error);
IF error <> 0 THEN commerr(error);
END ; { procedure send_stinger }

PROCEDURE decode_msg;
BEGIN
sting_az.b[2] := sting_io_in[1] ;
sting_az.b[1] := sting_io_in[2] ;
sting_el.b[2] := sting_io_in[3] ;
sting_el.b[1] := sting_io_in[4] ;
sting_stat := sting_io_in[7] ;
sting_az.i := (1440+sting_az.i) mod 2880;
sting_el.i := ((4320-TRUNC((sting_el.i*el_lsb)/az_lsb)) mod 2880)-1440;
END;


{--------------------------------------
{  Procedure stinger_manager
{   PURPOSE : determines azimuth and elevation from sting_io_in
{             calls hook_rack_process for track azimuth and elevation
{             computes azimuth and elevation and range error data
{   INPUT   : sting_io_in
{             hook_rec
{   OUTPUT  : az_error
{             el_error
{             in_range_bit
{             locked_on
{-------------------------------------}
PROCEDURE stinger_manager ;
BEGIN
decode_msg;
calibrate(sting_az.i,sting_el.i);
IF hook_a_track THEN
  BEGIN
  sting_hook ;
  az_error.i := TRUNC(hook_az/az_lsb) - sting_az.i;
  IF az_error.i > 1440 THEN az_error.i := az_error.i - 2880 ELSE
  IF az_error.i < -1440 THEN az_error.i := az_error.i + 2880 ;
  el_error.i := TRUNC(hook_el/az_lsb) - sting_el.i ;
```

```
  IF (hook_range <= sting_max_rng) AND (hook_range >= sting_min_rng) THEN
    in_range_bit := 8
   ELSE
    in_range_bit := 0 ;
  lockon:=(ABS(el_error.i)<=16) AND (ABS(az_error.i)<=16);
  send_stinger;
  END;
END ; { procedure stinger_manager }


(*------------------------------
(* Rx char driven state machine
(*------------------------------*)
PROCEDURE sting_handler ;
VAR  x : INTEGER ;
     by: BYTE ;
BEGIN
actual := OSREAD(serialid,stingfifo,statrec[8],error);
FOR x := 1 to statrec[8] DO
  BEGIN
  by := stingfifo[x] ;
  CASE sting_state OF
    0:IF by = syn1 THEN sting_state := 1 ;
                        (* also enable timout        *)
    1:IF by = syn2 THEN
        sting_state := 2
      ELSE
       sting_state := 0 ;
                        (* also disable timout       *)
    2:BEGIN
      sting_start := sting_start + 1 ;
      sting_io_in[sting_start] := by ;
      IF sting_start = sting_length THEN
        BEGIN
        sting_state := 0 ;
                        (* also disable timout       *)
        sting_start := 0 ;
        stinger_manager;
        END;
      END;
    END; (* case  *)
  END; (* for..next*)
END;


(*------------------------------
(* initialize 8274
(*------------------------------*)
PROCEDURE sting_init ;

BEGIN
commerr(998);
OSDELETEPROCESS(sp,error);
wrapup ;
(*--------------- attach the serial port --*)
```

```
pathname := devname ;
pathname[1] := CHR(7) ;
reserved := 0 ;
serialid := OSATTACH(pathname,oldfilemode,r served,updateaccess,error);
IF error <> 0 THEN commerr(error);
(*------------------ open serial -----------*)
OSOPEN(serialid,1,error);
IF error <> 0 THEN commerr(error);
(* ------------- init serial -------------- *)
parblock[1] := CHR(1) ;  (* mode *)
parblock[2] := CHR(1) ;  (* async *)
parblock[3] := CHR(8) ;  (* 8 dtat bits *)
parblock[4] := CHR(1) ;  (* 1 stop bits *)
parblock[5] := CHR(0) ;  (* nadda parity *)
OSSETSTATUS(serialid,parblock,5,error);
IF error <> 0 THEN commerr(error);
(*-------  new fifo ---------*)
fifo.mode := CHR(4);
NEW(fifo.ptr);
fifo.length := fifolength;
OSSETSTATUS(serialid,fifo,5,error);
IF error <> 0 THEN commerr(error);
(*--------------- baud rate -----------------*)
parblock[1] := CHR(7) ;  (* mode *)
parblock[2] := CHR(14) ; (* 9600 *)
OSSETSTATUS(serialid,parblock,2,error);
IF error <> 0 THEN commerr(error);
(*---------------- time out ------------*)
parblock[1] := CHR(2) ;
parblock[2] := CHR(0) ;
parblock[3] := CHR(1000) ;
OSSETSTATUS(serialid,parblock,3,error);
IF error <> 0 THEN commerr(error);
(*---------------- signals ------------*)
parblock[1] := CHR(60) ;
parblock[2] := CHR(28H) ;
parblock[3] := CHR(0) ;
OSSETSTATUS(serialid,parblock,3,error);
IF error <> 0 THEN commerr(error);
(*---------------- data mode ------------*)
parblock[1] := CHR(6) ;
OSSETSTATUS(serialid,parblock,1,error);
IF error <> 0 THEN commerr(error)
 ELSE
  BEGIN
  el_off_i := ROUND(el_off/az_lsb);
  sting_start := 0 ;
  sting_state := 0 ;
  sting_io_out[1] := syn1 ;
  sting_io_out[2] := syn2 ;
  status_bit := 16 ;
  led_bit := 7 ;
  in_range_bit := 0 ;
```

```
  commerr(999);
  send_stinger;
  poll_cnt:=poll_max;
  sp := OSFORKPROCESS(sting_it,20,TRUE,2000,error);
  END ;
END ;


(*-------------------------------
(* task used to poll serial fifo
(*-----------------------------*)
PROCEDURE sting_it ;
BEGIN
REPEAT
  OSDELAY(sting_delay);
  IF poll_cnt < poll_max THEN poll_cnt := poll_cnt + 1;
  WHILE char_det DO
    BEGIN
    sting_handler ;
    poll_cnt:=0;
    END;
  sting_on := (poll_cnt <> poll_max);
UNTIL FALSE ;
END ;
```

```
$NOLIST
$COMPACT(-c nst in c d -)

MODULE builtinradar ;

$INCLUDE ('w'incs'common.inc-text-)
$INCLUDE ('w'incs'conpas.inc-text-)
$INCLUDE ('w'incs'ospasprocs.inc-text-)
$INCLUDE ('w'incs'ospastypes.inc-text-)

(*-----------------------------------------*)
(* INTERFACE SECTION                       *)
(*-----------------------------------------*)

PUBLIC tablemgr ;
CONST       io_buff_lng = 11;
            pos_lsb   = 2.34375 ;          { TPQ coordinate units }
            vel_lsb   = 2.856   ;          { TPQ velocity units }

TYPE
            byte_int    =  RECORD
                             CASE BOOLEAN OF
                               TRUE : (b:ARRAY [1..2] of BYTE) ;
                               FALSE: (i:INTEGER);
                           END ; { record }
            clock_type  =  RECORD
                            CASE BOOLEAN OF
                              TRUE : (b:ARRAY [1..4] of BYTE) ;
                              FALSE: (i:LONGINT);
                           END ; { record }
VAR
            todclock    :  clock_type ;
            io_buffer   :  ARRAY [1..io_buff_lng] of BYTE ;


PUBLIC tablemgr ;
PROCEDURE msg_handler ;

PUBLIC BUILTINRADAR;
PROCEDURE activateradar;
PROCEDURE deactivateradar;

PRIVATE builtinradar ;

CONST       tvp_lsb   = 0.07616 ;

VAR         error   : WORD;
            rpid    : WORD;
            io_table: ARRAY[1..10,1..io_buff_lng] of BYTE;
            maneuver: BYTE;
(*--------------------------------------*)
(*  INITRADAR initializes io table      *)
(*--------------------------------------*)
```

```
PROCEDURE initradar;
VAR temp : byte_int ;
    junk : clock_type ;
    count: INTEGER ;
    parmtbl : ARRAY [1..10,1..6] of INTEGER;
BEGIN
maneuver := 0;
parmtbl[1,1]:=67;
parmtbl[1,2]:=15000;
parmtbl[1,3]:=10000;
parmtbl[1,4]:=150;
parmtbl[1,5]:=194;
parmtbl[1,6]:=194;
parmtbl[2,1]:=67;
parmtbl[2,2]:=14000;
parmtbl[2,3]:=9000;
parmtbl[2,4]:=150;
parmtbl[2,5]:=194;
parmtbl[2,6]:=194;
parmtbl[3,1]:=71;
parmtbl[3,2]:=15000;
parmtbl[3,3]:=1000;
parmtbl[3,4]:=50;
parmtbl[3,5]:=216;
parmtbl[3,6]:=216;
parmtbl[4,1]:=68+8;
parmtbl[4,2]:=1400;
parmtbl[4,3]:=2800;
parmtbl[4,4]:=50;
parmtbl[4,5]:=1;
parmtbl[4,6]:=1;
parmtbl[5,1]:=66+8;
parmtbl[5,2]:=-5000;
parmtbl[5,3]:=1500;
parmtbl[5,4]:=3000;
parmtbl[5,5]:=43;
parmtbl[5,6]:=200;
parmtbl[6,1]:=67+8;
parmtbl[6,2]:=15000;
parmtbl[6,3]:=-10000;
parmtbl[6,4]:=50;
parmtbl[6,5]:=194;
parmtbl[6,6]:=194;
parmtbl[7,1]:=67+16;
parmtbl[7,2]:=4000;
parmtbl[7,3]:=-8000;
parmtbl[7,4]:=150;
parmtbl[7,5]:=194;
parmtbl[7,6]:=194;
parmtbl[8,1]:=68+16;
parmtbl[8,2]:=1000;
parmtbl[8,3]:=-1000;
parmtbl[8,4]:=50;
```

```
parmtbl[8,5]:=230;
parmtbl[8,6]:=250;
parmtbl[9,1]:=68+24;
parmtbl[9,2]:=-1400;
parmtbl[9,3]:=-2800;
parmtbl[9,4]:=50;
parmtbl[9,5]:=1;
parmtbl[9,6]:=1;
parmtbl[10,1]:=66+24;
parmtbl[10,2]:=-500;
parmtbl[10,3]:=1500;
parmtbl[10,4]:=1500;
parmtbl[10,5]:=43;
parmtbl[10,6]:=200;
FOR count := 1 to 10 DO
  BEGIN
  io_table[count,1] := parmtbl[count,1];
  io_table[count,2] := count  ;
  temp.i := round(parmtbl[count,2]/pos_lsb) ;
  io_table[count,3] := temp.b[2] ;
  io_table[count,4] := temp.b[1] ;
  temp.i := round(parmtbl[count,3]/pos_lsb);
  io_table[count,5] := temp.b[2] ;
  io_table[count,6] := temp.b[1] ;
  temp.i := round(parmtbl[count,4]/pos_lsb);
  io_table[count,7] := temp.b[2] ;
  io_table[count,8] := temp.b[1] ;
  io_table[count,9] := parmtbl[count,5] ;
  io_table[count,10] := parmtbl[count,6] ;
  junk.i := todclock.i DIV 625 ;
  io_table[count,11] := junk.b[1] ;
  END;
END ; (* proc *)

(*-------------------------------------------*)
(* UPDATE update io table and send track data  *)
(*-------------------------------------------*)
PROCEDURE update ;
VAR count : INTEGER ;
    temp,temps  : byte_int ;
    tempv : INTEGER ;
    tl    : BYTE ;
    tclock,junk: clock_type;

BEGIN
FOR count := 1 to 10 DO
  BEGIN
  tclock.i := todclock.i ;
  junk.i := tclock.i div 625 ;
  tl := (junk.b[1] ) - io_table[count,11] ;
  temp.b[2] := io_table[count,3] ;
  temp.b[1] := io_table[count,4] ;
  IF io_table[count,9] >= 128 THEN
```

```
    tempv := io_table[count,9]-256
   ELSE
    tempv := io_table[count,9] ;
  temp.i := temp.i + TRUNC(tl*tempv*tvp_lsb);
  io_table[count,3] := temp.b[2] ;
  io_table[count,4] := temp.b[1] ;
  temp.b[2] := io_table[count,5] ;
  temp.b[1] := io_table[count,6] ;
  IF io_table[count,10] >= 128 THEN
    tempv := io_table[count,10]-256
   ELSE
    tempv := io_table[count,10] ;
  if count = 3 then
    begin
    tempv:=trunc(sin(maneuver)*40);
    io_table[count,10]:=tempv;
    maneuver:=maneuver+1;
    end;
  temp.i := temp.i + TRUNC(tl*tempv*tvp_lsb);
  io_table[count,5] := temp.b[2] ;
  io_table[count,6] := temp.b[1] ;
  io_table[count,11] := junk.b[1] ;
  END ;(* for .. next *)
FOR count := 1 to 10 DO
  BEGIN
  for tl := 1 to io_buff_lng do io_buffer[tl]:=io_table[count,tl];
  msg_handler ;
  END;
END ; (* proc *)

(*----------------------------*)
(* SENDTIME sends time message *)
(*----------------------------*)

PROCEDURE send_time ;
VAR x : INTEGER ;
    tclock:clock_type;
BEGIN
tclock.i:=todclock.i;
io_buffer[1]:=128;
io_buffer[2]:=0;
for x := 4 downto 1 DO
   io_buffer[7-x]:=tclock.b[x];
for x := 7 to 11 do
   io_buffer[x]:=0;
msg_handler;
END ;(* proc *)

(*----------------------------*)
(* RADARSTART task to simulate *)
(*           updated track msg*)
(*----------------------------*)
```

```
PROCEDURE radarstart;
BEGIN
initradar;
REPEAT
    OSDELAY(1500);
    update ;
    send_time ;
UNTIL FALSE ;
END ;

(* ----------------------
(*  activate the radar
(* ----------------------*)
PROCEDURE activateradar;
BEGIN
rpid:=OSFORKPROCESS(radarstart,40,TRUE,2000,error);
END;


(* ----------------------
(*  deactivate the radar
(* ----------------------*)
PROCEDURE deactivateradar;
BEGIN
OSDELETEPROCESS(rpid,error);
END;
```

```
$COMPACT
alert:DO;

/*  audible alarm   */

beep:PROCEDURE PUBLIC;

DECLARE spkr BYTE AT (0DFE42H);
DECLARE i    WORD;

  DO i = 1 to 150;
    spkr = 255;
    call time(2);
    spkr = 0;
    call time(2);
    END;

END beep;

/*  no interrupt 0  */

zint:PROCEDURE INTERRUPT 0 PUBLIC;
END zint;


/*  init 8087  */

zeroinit:PROCEDURE PUBLIC;

   call init$real$math$unit;
   call set$interrupt(0,zint);
END zeroinit;
END alert;
```

CLAIMS

What is claimed is:

1. A target acquisition and cueing system having a first sensor (2) for determining the position of targets and a display (26) for displaying the positions of those targets relative to the location of a weapon system (34), characterized in that:

the weapon system (34) is remotely located from the first sensor (2);

a second sensor (70) determines the pointing direction of the weapon system (34);

a computing means (24) receives from the first sensor (2) the location of the targets and compares the location of the weapon system (34) thereto to calculate the relative azimuth, elevation, and range of each of the targets from the weapon system (34), and receives from the second sensor (70) the pointing direction of the weapon system (34) and calculates the errors between the pointing direction of the weapon system (34) and the position of the targets;

the display means (26) displays the calculated relative azimuths, elevations, and ranges of the targets from the weapon system (34);

a selection means (28) designates one of the displayed targets (6); and

a sight means (32) displays to the weapon system (34) the errors calculated by the computing means (24) between the pointing direction of the weapon system (34) and the location of the designated target (6).

2. The target acquisition and cueing system according to Claim 1 further characterized in that the sight means (32) also displays to the weapon system (34)

an indication of whether the designated target (6) is within a predetermined range of the weapon system (34).

3. The target acquisition and cueing system according to any preceding claim further characterized in that the display means (26) includes multiple displays, one of which comprises a situation display (78) showing a graphic presentation of the targets sensed by the first sensor means (2) in relation to the weapon system (34) and another of which comprises a parameter display (80) showing numerical data relating to characteristics of a selected target relative to the position of the weapon system (34).

4. The target acquisition and cueing system according to any preceding claim further characterized in that the computing means (24) calculates an alert range representing a predetermined range from the weapon system (34) and provides an audible alert when a target has been calculated to have entered the alert range.

5. The target acquisition and cueing system according to any preceding claim further characterized in that the weapon system (34) comprises a portable launcher.

6. The target acquisition and cueing system according to any preceding claim further characterized in that the computing means (24) comprises a portable computer.

7. The target acquisition and cueing system according to any preceding claim further characterized in that it comprises a communication system located between

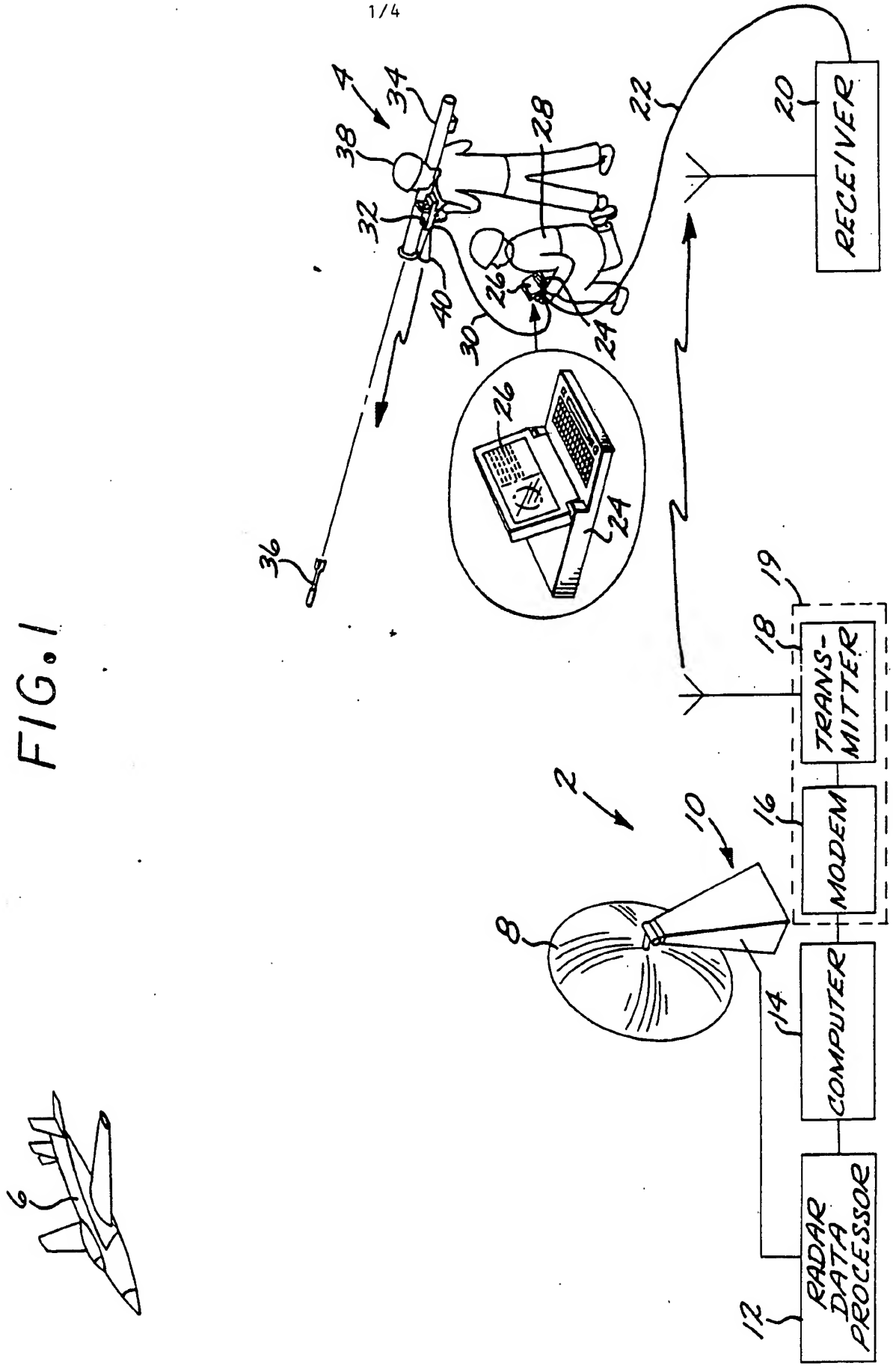
the first sensor (2) and the computing means (24) having an encoder (16) for encoding the locations of the targets into a digital message; a frequency modulator for frequency modulating the digital message; a transmitter (18) which transmits the frequency modulated message; a receiver (20) which receives the frequency modulated message; and a demodulator which demodulates the message into a digital message to be received by the computing means (24).

8. The target acquisition and cueing system according to any preceding claim further characterized in that a human operator (38) uses the sight (32) in launching the weapon (36) from the weapon system (34).

9. The target acquisition and cueing system according to any preceding claim further characterized in that a human operator (28) uses the display (26) to prioritize displayed targets and designate one of the targets (6).

10. The target acquisition and cueing system according to any preceding claim further characterized in that the display (26) is located at a command center which is remote from the weapon system (34) and the first sensor (2) and which is in communication with multiple weapon systems.

FIG. 1


RADAR DATA PROCESSOR
COMPUTER
MODEM
TRANS-MITTER
RECEIVER
12
14
16
18
19
20
22
24
26
28
30
32
34
36
38
40
2
4
6
8
10

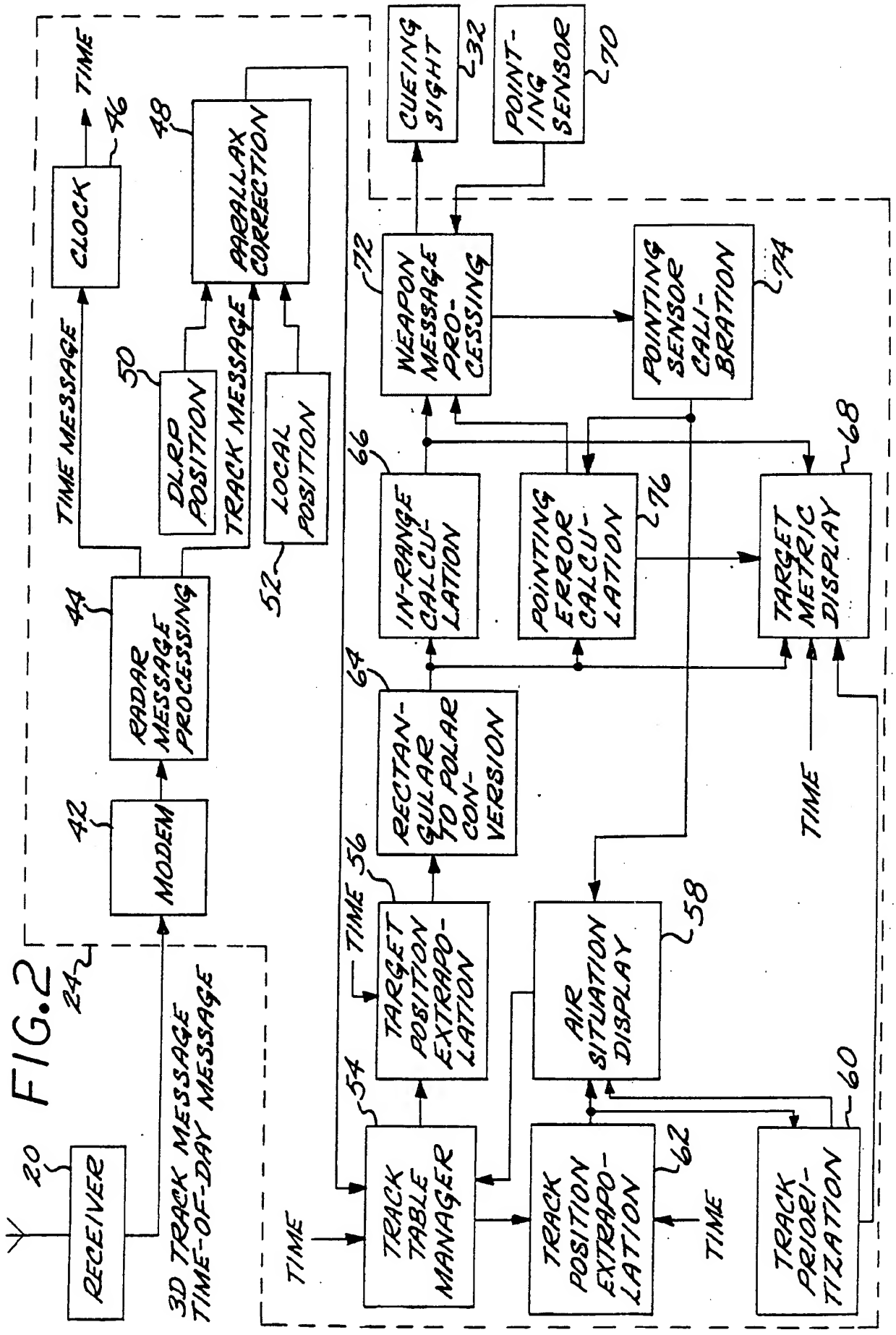

20
RECEIVER
3D TRACK MESSAGE
TIME-OF-DAY MESSAGE
24
MODEM
42
RADAR MESSAGE PROCESSING
44
TIME MESSAGE
TRACK MESSAGE
CLOCK
46
TIME
PARALLAX CORRECTION
48
DLRP POSITION
50
LOCAL POSITION
52
CUEING SIGHT
32
POINTING SENSOR
70
WEAPON MESSAGE PROCESSING
72
POINTING SENSOR CALIBRATION
74
IN-RANGE CALCULATION
66
POINTING ERROR CALCULATION
76
TARGET METRIC DISPLAY
68
RECTANGULAR TO POLAR CONVERSION
64
TARGET POSITION EXTRAPOLATION
56
TIME
AIR SITUATION DISPLAY
58
TRACK TABLE MANAGER
54
TRACK POSITION EXTRAPOLATION
62
TRACK PRIORITIZATION
60
TIME
TIME
TIME

FIG.2

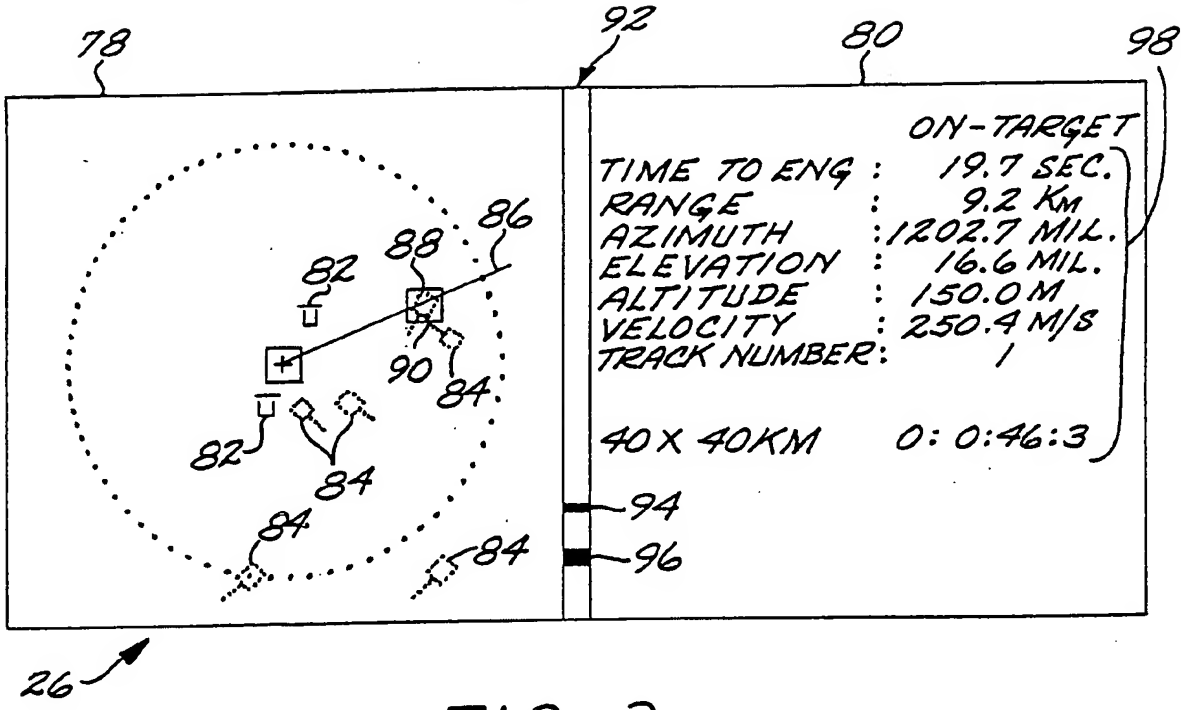


FIG. 3

100

```
***GRID DISPLAY UNIT PARAMETER MENU***
DATA LINK REFERENCE POINT(METER) N:0  E:0  ALT:0
FIRE UNIT LOCATION (METERS)      N:0  E:0  ALT:0
MAGNETIC NORTH DECLINATION(MILS) E:0
ALERT RANGE (METERS)              :15000
AUDIBLE ALERT (0-OFF/1-ON)        :0
AUTOMATIC HOOK (0-OFF/1-ON)       :0
DEMO MODE (0-OFF/1-ON)            :0
PRIORITY (0-OFF/1-ON)             :0
PRIORITY SECTOR   STARTING MILS   :0
PRIORITY SECTOR   ENDING MILS     :0
VELOCITY CUTOFF                   :0
ALTITUDE CUTOFF                   :0
**<ESC>TO RETURN TO MONITOR**
```

FIG. 4

FIG. 5

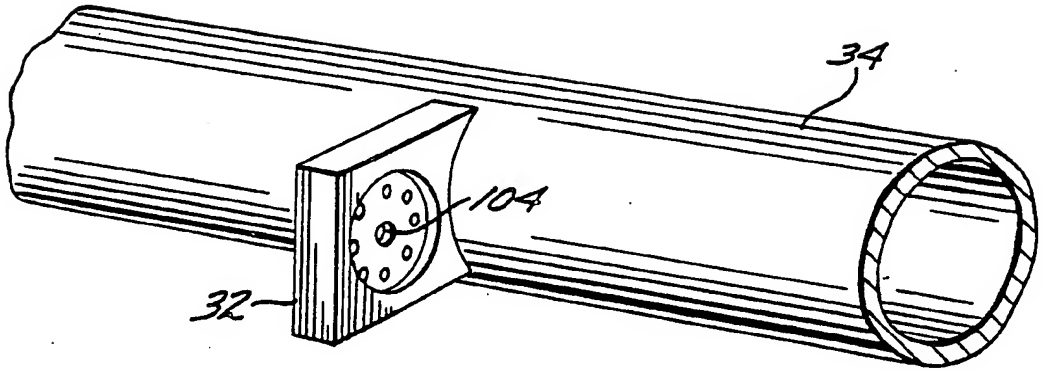
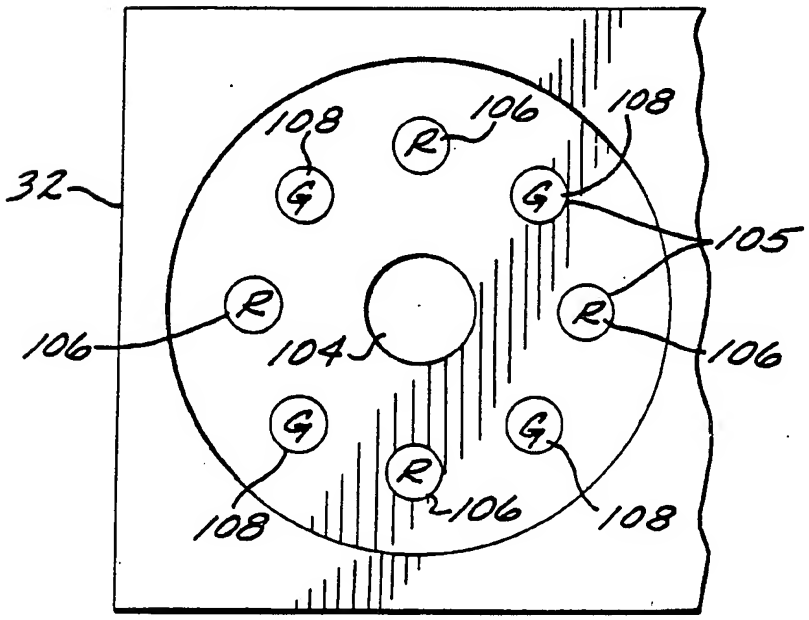


FIG. 6



INTERNATIONAL SEARCH REPORT

International Application No PCT/US 87/02435

I. CLASSIFICATION OF SUBJECT MATTER (if several classification symbols apply, indicate all) [6]

According to International Patent Classification (IPC) or to both National Classification and IPC

IPC[4]: F 41 G 5/08; F 41 G 3/14

II. FIELDS SEARCHED

Minimum Documentation Searched [7]

Classification System	Classification Symbols
IPC[4]	F 41 G; G 01 S

Documentation Searched other than Minimum Documentation to the Extent that such Documents are included in the Fields Searched [8]

III. DOCUMENTS CONSIDERED TO BE RELEVANT [9]

Category *	Citation of Document, [11] with indication, where appropriate, of the relevant passages [12]	Relevant to Claim No. [13]
X	Revue Internationale de Défense, volume 17, no. 6, 1984, (Cointrin-Geneva, CH), M. Hewish: "Le système léger de conduite de tir de l'artillerie Quickfire", pages 752-753 see the whole document --	1,2,5-10
A	Revue Internationale de Défense, volume 10, no. 3, 1977, (Cointrin-Geneva, CH), R. Meller: "Le système de conduite de tir tous temps Flycatcher pour canons antiaériens et missiles sol-air de défense rapprochée", pages 495-499 see pages 496-499, paragraph: "Description du système" et "Le processus de mise en œuvre" --	1-3,5,9,10
A	FR, A, 2415285 (BOFORS) 17 August 1979 see the whole document --	1,8-10
A	US, A, 4267562 (P.K. RAIMONDI) 12 May 1981 -- ./.	

* Special categories of cited documents: [10]

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

IV. CERTIFICATION

Date of the Actual Completion of the International Search	Date of Mailing of this International Search Report
20th January 1988	16 MAR 1988
International Searching Authority	**Signature of Authorized Officer**
EUROPEAN PATENT OFFICE	P.C.G. VAN DER PUTTEN

III. DOCUMENTS CONSIDERED TO BE RELEVANT (C NTINUED FR M THE SECOND SHEET)

Category *	Citation of Document, with indication, where appropriate, of the relevant passages	Relevant to Claim No
A	WO, A, 80/00618 (ERICSSON) 3 April 1980 --	
A	Proceedings of the IEEE 1986 National Aerospace and Electronics Conference NAECON 1986, Dayton Convention Center, 19-23 May 1986, volume 4, D.A. Dotson et al.: "Comparison of techniques for "hooking" of tracks on a short-range air defense command and control (Shorad C^2) display", pages 1072-1075 --------	

ANNEX TO THE INTERNATIONAL SEARCH REPORT ON INTERNATIONAL PATENT APPLICATION NO. US 8702435

SA 19078

This annex lists the patent family members relating to the patent documents cited in the above-mentioned international search report. The members are as contained in the European Patent Office EDP file on 17/02/88
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

Patent document cited in search report	Publication date	Patent family member(s)		Publication date
FR-A- 2415285	17-08-79	NL-A-	7900256	20-07-79
		GB-A,B	2012925	01-08-79
		DE-A-	2901873	19-07-79
		US-A-	4266463	12-05-81
		SE-B-	420766	26-10-81
		SE-A-	7800577	19-07-79
US-A- 4267562	12-05-81	None		
WO-A- 8000618	03-04-80	EP-A-	0022783	28-01-81
		SE-B-	421250	07-12-81
		SE-A-	7809551	13-03-80
		US-A-	4458246	03-07-84